Die approbierte Originalversion dieser Dissertation ist in der Hauptbibliothek der Technischen Universität Wien aufgestellt und zugänglich.
http://www.ub.tuwien.ac.at

The approved original version of this thesis is available at the main library of the Vienna University of Technology.

http://www.ub.tuwien.ac.at/eng

TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

DISSERTATION

Development and Simulation Framework for Industrial Production Systems

Ausgeführt zum Zwecke der Erlangung des akademischen Grades eines
Doktors der technischen Wissenschaften (Dr.techn.)

unter der Leitung von
Univ.–Prof. Dipl.–Ing. Dr.sc.techn. Georg Schitter
Institut für Automatisierungs– und Regelungstechnik (E376)

eingereicht an der
Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik

von
Ingo Hegny
Matrikelnummer: 9725162
Hauptstrasse 3, 3435 Zwentendorf an der Donau, Österreich

Zwentendorf an der Donau, im Februar 2014

Erster Gutachter:	Univ.–Prof. Dipl.-Ing. Dr.sc.techn. Georg Schitter
Zweiter Gutachter:	Prof. Dr. Luca Ferrarini (Politecnico di Milano, Italien)

Abstract

Production systems are complex structures. The growing need for customized mass-goods and decreasing life cycles of consumer goods imposes the demand for flexible manufacturing systems and frequent changes to deployed plants. Different disciplines are involved in their creation. More and more functionalities are covered by automatic control software. Hence, its share in terms of costs and development time is ever increasing. However, in prevalent, serialised development workflows, automatic control development is one of the last disciplines involved. This imposes pressure regarding functionality, costs, and also commissioning time on automatic control engineers. Model-based engineering is a promising approach to improve this situation as it enables and fosters the collaboration of all involved disciplines.

The aim of this thesis is to introduce a new multi-disciplinary approach for the specification, development, and validation of production systems. The core of the proposed approach is the modelling infrastructure, which is inspired by the Model Driven Architecture (MDA). A set of models, each for different engineering aspects, is linked to provide coherent, multi-disciplinary data that is relevant for the implementation of production systems. Functional modules, so called Automation Components, are introduced. These are the building blocks of automated production systems. Furthermore, they facilitate the specification and implementation process, as the re-use of Automation Components reduces the engineering effort. Furthermore, once specified and implemented components usually have higher quality.

The proposed workflow targets the simultaneous system specification in all involved disciplines. Currently used engineering tools, such as CAD tools or PLC programming tools, shall be integrated and model-transformation facilities are used to extract engineering data. The availability of a comprehensive specification early in the engineering workflow allows the parallelized implementation in the involved disciplines (e.g. mechanical construction, programming of the automatic control system). Hence, the overall development and implementation time can be reduced. For further reconfiguration activities it is important to keep the specification of the automated system accurate and coherent with the actual implementation. The hierarchical aggregation of Automation Components, with clearly defined interfaces and encapsulated behaviour, support this task, as changes are locally confined.

The direct inclusion of plant behaviour simulation in the specification and development workflow is an innovation. This allows also to keep the simulation model of the automated plant coherent with the actual (or planned) plant configuration. Implementation can already be started without the physical availability of the controlled plant. The simulation environment provides a virtual representation of the plant and allows the validation of automatic control applications (i.e. virtual commissioning).

The simulation framework is based on the IEC 61499 compliant runtime environment FORTE. Its event based execution semantics is suitable for discrete event simulation. Also the modular aggregation of functionality—based on Function Blocks—facilitates the modeling of modular plant models (with the same structure as the Automation Components). The automatic control application and the plant simulation application are executed on the same automatic control system (one or multiple automatic control devices). First, coupling plant simulation and automatic control is possible without communication overhead. Second, the execution of the automatic control application is not restricted, as it is the same type of runtime environment they are deployed to later in the field. Different scenarios, such as full simulation, hybrid simulation, inclusion of external simulation tools, and finally operation and testing, support automatic control engineers during various phases of the automatic control development.

Finally, test cases from three different domains are selected. Discrete manufacturing plants, robotics applications, and process technology are covered. Each domain has unique requirments that have to be fulfilled by the proposed engineering process. For the evaluation of the proposed approach all relevant elements of the workflow are implemented. The feasibility of an integrated engineering and simulation workflow is validated with the help of these three test cases. Different models and interfaces (between the disciplines) are maintained through the collaboration of experts from multiple disciplines. Automatic control engineers gain access to the well-suited validation tool of discrete event simulation with little additional specification effort by reusing specification data for the automatic generation of simulation models. This helps to reduce the unproductive, and thus costly, commissioning and ramp-up time. With the establishment of comprehensive libraries of Automation Components the engineering effort, across all involved disciplines, is significantly reduced. Hence, the engineering cycles for the creation of production systems or their adaptation to meet new requirements are shortened.

Kurzfassung

Produktionssysteme sind komplexe Strukturen. Der steigende Bedarf an individualisierten Produkten und die gleichzeitige Reduktion der Lebenszyklen von Konsumgütern erfordern Adaptionen der Fertigungssysteme. Deshalb sind häufige Änderungen an den eingesetzen Produktionsanlagen zu erwarten.

An ihrer Entstehung wirken unterschiedliche Disziplinen (z.B. Mechanik, Elektrik, Pneumatik, Steuerungstechnik) mit. Immer mehr Funktionen, die früher mechanisch oder elektromechanisch bereitgestellt wurden, werden nun durch Steuerungssysteme übernommen. Daher steigt auch der Anteil der Steuerungstechnik an den Gesamtkosten stetig. In häufig eingesetzen Entwicklungsworkflows, die eine serielle Bearbeitung vorsehen, ist der steuerungstechnische Entwurf der letzte Entwicklungsschritt vor der Inbetriebnahme. Daher lastet hoher Druck auf den Entwicklerinnen und Entwicklern der Steuerungssysteme, sowohl in Bezug auf Funktionalität, als auch bezüglich Kosten und Inbetriebnahmetermin. Modell-basierte Entwicklung ist ein vielversprechender Ansatz, um eine kooperative Entwicklung aller involvierten Disziplinen zu ermöglichen.

Das Ziel dieser Arbeit ist es, einen neuen, multi-disziplinären Ansatz für die Spezifikation, Entwicklung und Validierung von Produktionsanlagen zu entwickeln. Die Infrastruktur zur Modellierung, angelehnt an die Model Driven Architekture (MDA), ist der zentrale Punkt des vorgeschlagenen Ansatzes. Eine Reihe von Modellen, die jeweils unabhängig voneinander Platz für unterschiedliche Aspekte aus den verschiedenen Domänen bieten, wird spezifiziert. Zur Erhöhung der Wiederverwendbarkeit und zur Vereinfachung der Spezifikation und Implementierung wurden funktionelle Module, sogenannte Automationskomponenten (*Automation Components*), eingeführt. Diese sind die Bausteine für den Aufbau von automatisierten Produktionssystemen.

Der vorgeschlagene Entwurfsprozess zielt auf eine konzentrierte, simultane Systemspezifikation aller beteiligten Disziplinen ab. Derzeit verwendete Entwurfswerkzeuge (z.B. CAD Programme, Steuerungsentwicklungs-

umgebungen) sollen integriert werden und Modell-Transformationen sollen zur automatischen Extraktion von Entwurfsdaten genutzt werden Die frühe Verfügbarkeit umfassender Spezifikationsdaten ermöglicht eine weitgehend parallelisierte Implementierung in allen Disziplinen (z.B. mechanische Konstruktion, Steuerungsentwicklung). Dadurch kann eine Reduktion der Entwicklungs- und Implementierungszeit erreicht werden. Auch für spätere Rekonfigurations-Tätigkeiten ist es wichtig, die Spezifikation und die tatsächliche Anlagenkonfiguration synchron zu halten. Die hierarchische Aggregation der Automationskomponenten mit ihren klar definierten Schnittstellen und gekapseltem Verhalten unterstützt diese Aufgabe, da Änderungen rein lokal beschränkt sind.

Die direkte Einbeziehung der Verhaltenssimulation der Anlage in den Spezifizierungs- und Entwicklungsprozess von Produktionsanlagen ist eine Innovation. Dies erlaubt die Simulationsmodelle der automatisierten Anlagen synchron mit den tatsächlichen (oder geplanten) Anlagenkonfiguration zu halten. Die Implementierung des Steuerungssystems kann bereits ohne die physische Verfügbarkeit der projektierten Anlage begonnen werden.

Das vorgeschlagene Simulations-Framework basiert auf der IEC 61499-konformen Laufzeitumgebung FORTE. Durch ihre ereignis-basierte Ausführungssemantik ist sie für diskrete Ereignissimulation geeignet. Auch die modulare Aggregation von Funktionalitäten, auf Basis von Funktionsbausteinen, erleichtert die Modellierung von modularen Anlagen und Komponenten. Steuerungsanwendungen und die Anlagensimulation wird auf dem selben Steuerungssystem (eine oder mehrere Steuerungen) ausgeführt. Dies erlaubt einerseits eine einfache Kopplung von Simulation und Steuerungsanwendung ohne Kommunikationsoverhead. Andererseits wird die Ausführung der Steuerungsanwendung nicht beschränkt, da das selbe Steuerungssystem (inkl. Laufzeitsystem) wie im Betrieb verwendet wird. Verschiedene Szenarien, von der vollständigen Simulation, über die hybride Simulation, der Einbeziehung externer Simulationsanwendungen bis hin zu Tests und operativem Betrieb, unterstützen den Steuerungstechniker in den verschiedenen Phasen der Entwicklung von Steuerungsanwendungen.

Zur Validierung des vorgestellten Ansatzes werden Testfälle aus drei verschiedenen Bereichen vorgestellt. Diskrete Fertigungsanlagen, Robotik-Anwendungen und Prozesstechnik werden abgedeckt. Jede Domäne hat eigene Anforderungen, die vom vorgestellten Entwicklungsprozess erfüllt werden müssen. Für die Evaluierung des vorgeschlagenen Ansatzes werden alle relevanten Elemente des Workflows implementiert. Die Machbarkeit eines integrierten Entwurfs- und Simulationsprozesses wird anhand dieser drei Testfälle gezeigt. Die Modellierungsinfrastruktur unterstützt eine klare Aufgabenteilung. Verschiedene Modelle und Schnittstellen (zwischen den Disziplinen) werden kooperativ von Expertinnen und Experten aus unterschiedlichen Disziplinen spezifiziert. Das gut geeignete Validierungswerkzeug der

diskreten Ereignissimulation wird durch die Wiederverwendung von Spezifikationsdaten zur automatischen Erstellung der Simulationsmodelle ohne großen Zusatzaufwand erschlossen. Dies ermöglicht unproduktive, und daher kostenintensive, Inbetriebnahmephasen zu verkürzen. Mit dem Aufbau von umfangreichen Bibliotheken von Automationskomponenten kann der Entwicklungsaufwand aller involvierten Disziplinen signifikant reduziert werden. Daher können Entwurfszyklen für den Aufbau neuer sowie den Umbau bestehender Produktionssysteme verkürzt werden.

Acknowledgements

First of all I would like to thank Professor Georg Schitter for taking over the supervision of my thesis. Thank you also for encouraging me to persue high scientific standards.

I gratefully acknowledge the help of Professor Luca Ferrarini from the Politecnico di Milano for co-supervising this work and for many fruitful discussions during our common project. Mille grazie!

I would also like to thank my former collegues at the Automation and Control Institute at Vienna University of Technology. Working with you was inspiring and I really enjoyed working with you. Thank you for many fruitful discussions and your support. Especially I would like to mention Christoph Sünder, Monika Wenger, Martin Melik-Merkumians, Dr. Munir Merdan, Prof. Gerfried Zeichen, and Gabriele Grabensteiner.

Above all I would like to thank Dr. Alois Zoitl. You encouraged me to start a scientific career and were always supportive during so many years.

Furthermore, I like to express my gratitude to Dr. Thomas Strasser, Gerhard Ebenhofer, Dr. Martijn Rooker, Alessio Dede, Laurence Picci, Giuseppe Fogliazza, and Prof. Marco Colla for the interesting discussions we had in our collaborative projects.

For their effort to proof-read this thesis I want to thank Dr. Michael Fischer and Kerstin Zimmermann.

I would also like to acknowledge the support received from the funded projects KASA, MEDEIA, and MASGrid, which allowed me to conduct my research and broaden my horizon.

Finally, I would like to thank my family, who has been supportive during all the years of my studies. Herzlicher Dank an Helga, Peter und Ralf!

INGO HEGNY

Contents

List of Figures

List of Tables

List of Listings

List of Terms

AC Automation Component.

ACIM Automation Component Implementation Model.

ACM Automation Component Model.

API application programming interface.

BehM Behaviour Model.

BFB Basic Function Block.

BNF Backus-Naur Form.

BoM Bill of Material.

CAD Computer Aided Design.

CAE Computer Aided Engineering.

CAEX Computer Aided Engineering Exchange.

CAN Controller Area Network.

CBD Component-Based Development.

CBSE Component-Based Software Engineering.

CFB Composite Function Block.

CIM Computer Integrated Manufacturing.

CIM Computation Independent Model.

CNC Computer Numerical Control.

COLLADA COLLAborative Design Activity.

CORBA Common Object Request Broker Architecture.

COTS commercial-of-the-shelf.

DCS Distributed Control System.

DDES Distributed Discrete Event Simulation.

DiagM Diagnostics Model.

DOF degrees-of-freedom.

DOM Document Object Model.

DSL Domain Specific Language.

DSML Domain Specific Modeling Language.

ECC Execution Control Chart.

EMF Eclipse Modelling Framework.

ESFB Event Source Function Block.

ESM Execution System Model.

FB Function Block.

FBD Function Block Diagram.

FBN Function Block Network.

FDCML Field Device Configuration Markup Language.

FIPA Foundation for Intelligent Physical Agents.

FMS Flexible Manufacturing System.

HAL hardware abstraction layer.

HIL Hardware-in-the-Loop.

HMS Holonic Manufacturing System.

IC integrated circuit.

ICP Instrumentation- and Control-Point.

IEC 61499 IEC 61499 Function Blocks.

IL Instruction List.

IMS Intelligent Manufacturing System.

IP Intellectual Property.

IPC Industrial PC.

IPP Industrial Process Parameters.

JT Jupiter Tesselation.

LD Ladder Diagram.

M2C Model-to-Code.

M2M Model-to-Model.

M2T Model-to-Text.

MAS Multi-Agent System.

MDA Model-Driven Architecture.

MDSD Model Driven Software Development.

MECE MEDEIA Engineering and Configuration Environment.

MIC Model-Integrated Computing.

MM Mapping Model.

MMP massively parallel processing.

MWE Modeling Workflow Engine.

OMG Object Management Group.

OOP Object-Oriented Programming.

PAC Programmable Automation Controller.

PDES Parallel Discrete Event Simulation.

PIM Platform Independent Model.

PlaM Plant Model.

PLC Programmable Logic Controller.

PlC Plant Component.

PLM Product Lifecycle Management.

PM Platform Model.

PSM Platform Specific Model.

RIL Reality-in-the-Loop.

RQ research question.

SFC Sequential Function Chart.

SIFB Service-Interface Function Block.

SIL Software-in-the-Loop.

SMP symmetric multiprocessing.

ST Structured Text.

Sub-App Sub-Application.

UML Unified Modeling Language.

WCET Worst Case Execution Time.

XML Extensible Markup Language.

CHAPTER 1

Introduction

Manufacturing industries are facing many challenges today. On the one hand, labour-intensive production is moved to emerging markets. On the other hand, customer demands and requirements with respect to quantity—mass-customization down to lot-size one production—, quality, and delivery time are hard to fulfil with currently deployed rigid plant structures. To cope with this situation European industries are forced to install and operate flexible and cost effective production systems. Such flexible manufacturing systems are able to adapt to these future requirements [1]. Adaptivity of production systems requires changeability of the mechanical structures of the plant [2]. However, changes to the mechanical setup also infer changes of the electrical setup and the automatic control software application [3].

The automation and control of production plants take a huge amount of development time and costs. In the automotive sector these software and system related tasks currently make up about 55% of the total costs [4]. These costs are predicted to rise even higher. Especially in the industrial sector, the reusability of software is a crucial, yet not well addressed point. Often previously used automation software is used as a template, which is adapted to the new plant. The monolithic structure and the absence of a systematic approach, however, often lead to sub-optimal automatic control systems. Furthermore, system integrators and plant operators are often forced to use engineering tools from multiple device vendors, since many of today's production plants consist of multi-vendor devices. Each plant or system can be seen as a prototype.

A component and model based approach will result in a higher level of quality. Experts from multiple domains (e.g. electrical, mechanical, and control engineers) are involved in the development process. In the currently used development approaches friction losses between the domains due to unclear or unspecified interfaces impose a high risk to keep schedules and budgets. The composition of a multi-domain development team (and tool), already starting in the specification phase, imposes a higher effort at the beginning, but bears the chance to reduce the overall project time and to implement a solution that is closer to a global optimum [5]. Diverse methodologies and implementation technologies have to be integrated in order to enable an efficient engineering cycle of industrial automation and control systems.

The overall efficiency can also be increased by reducing the time for commissioning and ramp up. This also helps to reduce down-times during reconfiguration tasks. Simulation is well accepted as validation methodology in multiple engineering disciplines. However, currently there exist no simulation frameworks that allow to simulate the manufacturing system including the automatic control system. Hence, a simulation framework on the basis of an industrial automatic control environment is presented that allows to validate both the behaviour of the plant as well as the automatic control application.

1.1 Background

> *"The market increasingly demands products that are customised, yet available with short delivery times. Urgent attention to knowledge engineering is necessary to decrease time-to-market and cut delivery delays. Furthermore, 'rapid manufacturing' technologies will be needed to manufacture customised products." Manufuture High-Level Group* [1]

The production paradigm in western industrial countries is currently changing. Plants for mass production of goods are to be replaced by plants which allow mass customization or even the production with lot size one. The produced goods as well as the production process have to fulfil requirements demanded by the customers. The strategic research agenda of the Manufuture High-Level Group [1] states that manufacturing systems have to become *"adaptive, digital, networked, and knowledge based"* in order to cope with the imposed requirements (e.g. small lot sizes, short lead time, reliable production systems).

1.1.1 Manufacturing Systems Concepts

Many concepts for manufacturing systems have been developed and deployed since the beginning of industrial production [6, 7, 8]. The central parts of manufacturing systems are the production machines, including tools and fixtures. But also material handling and human workers (operating and managing the system) are important factors in manufacturing systems [6]. In modern manufacturing systems computer based control and management systems are deployed to support the human operators [9].

Splitting the production process into (small) tasks is a principal paradigm of manufacturing systems [7, 10]. Efficiency of the production process can be significantly increased, because single workstations can be more specialized on performing these tasks. Semi-finished goods have to be transferred between the involved workstations. In the early days of manufacturing the workstations as well as the material handling have been performed by human workers [6].

The first step towards automated manufacturing systems has been the introduction of automated material handling systems, for example conveyor belts. The manual assembly line, consisting of automated material handling systems and workstations, operated by human workers, brought a boost in efficiency [6]. These highly specialised manufacturing systems enabled the efficient, high-volume production of identical goods. Henry Ford's well known quotation is a good indicator on the rigid structure of such dedicated manufacturing systems:

> *"Any customer can have a car painted any colour that he wants so long as it is black." Henry Ford* [11]

As next step also the workstations have been automated. At the cost of flexibility the throughput of the machines has been further increased. These automated workstations, also called machine cells, have been optimized for the goods that shall be produced. They are the central elements of automated, dedicated manufacturing systems. If the high-volume production of goods, already known at the design time of the production system, is the aim, dedicated manufacturing systems are and will remain the most efficient choice.

The Flexible Manufacturing System (FMS) concept—introduced in the 1960s—was the first manufacturing system to integrate information technology [6]. Based on the Computer Numerical Control (CNC), it was the first manufacturing system which could cope with the challenges of product variety. It allows the machining of different products in medium volume on the same system. Highly sophisticated machine cells and a material handling system, which allows the transport of goods—in most cases from every machine cell to any other machine cell—provide this flexibility [6]. The hardware in such FMS is fixed. Also the hardware-related software (e.g. runtime environments, firmware) is fixed. However, product-related software (e.g. CNC programs) may be changed. Thus job scheduling can be modified and the machining of new parts is enabled. An advantage of the fixed hardware is the fast changeover, since only software has to be replaced. But it is also obvious that the mechanical system has to provide more functions than needed for the products targeted during the commissioning. Otherwise usage scenarios in which unpredicted products have to be machined would not be possible. This strives for complex machines, comprising of expensive hardware and control systems. Due to the fixed setup of the FMS it is not possible to replace or update it only partly. The complexity of the machines requires complex programs, which are hard to maintain. Therefore the deployed FMS are *"quite unreliable"* [3].

In the 1990s the trend to mass customization urged the need for new manufacturing systems. The Reconfigureable Manufacturing System approach aims at the flexibility of the production system. If new functionalities or process technologies are needed for the machining of the new product, basic process modules can be added, removed or upgraded [3]. These modules comprise hardware and software. With intrinsic continuous change in the Reconfigureable Manufacturing System concept, the system is *"open ended"* [3]. Thus the current setup of the system is well suited for the currently produced goods, which it has been adapted for.

Another component based approach is the Holonic Manufacturing System (HMS) concept [12, 13]. The HMS was developed within the Intelligent Manufacturing System (IMS)[1] program [14]. The main focus of the HMS concept lies on a new control architecture that is based on a modern software architecture. Modularisation and component based design breaks with the predominant monolithic software solutions for industrial automation [13]. The modularity of the software increases the maintainability. Clearly defined interfaces allow the modification or replacement of modules without affecting the other modules. To increase reusability the software modules are chosen to represent the physical components that comprise the production system [14]. The unity of hardware and software fostered the use of Distributed Control Systems (DCSs). Automatic control functionalities are executed collaboratively by multiple communicating entities. Such DCSs that allow to integrate automatic control functionality directly into the physical components, build the basis for the holonic architecture. Proponents of the HMS concept were also involved in the development of the international standard on distributed control—IEC 61499 Function Blocks (IEC 61499) [13]. On top of this DCS a self-organising, service oriented layer is used. For this purpose Multi-Agent Systems (MASs), based on standards introduced by the Foundation for Intelligent Physical Agents (FIPA)[2], are proposed [14]. Software Agents—the building blocks of MAS—are autonomous, interacting and flexible software components [15]. Multiple research groups are currently investigating the applicability of the proposed multi-layer approach with MAS. Merdan et al. [16, 17] provide evidence for the feasibility of the HMS approach for intra-logistic systems. Leitao et al. [18] show the applicability of MAS based control systems for manufacturing machines. Colombo et al. [19] provide a transition path to

[1]For more information on the IMS research and development program see http://www.ims.org (last access: 04.09.2012).

[2]FIPA was founded in 1996 as an independent organisation. Since 2005 FIPA is part of the IEEE Computer society and one of its standardisation committees. For more information on FIPA or its standards visit http://www.fipa.org (last access: 04.09.2012).

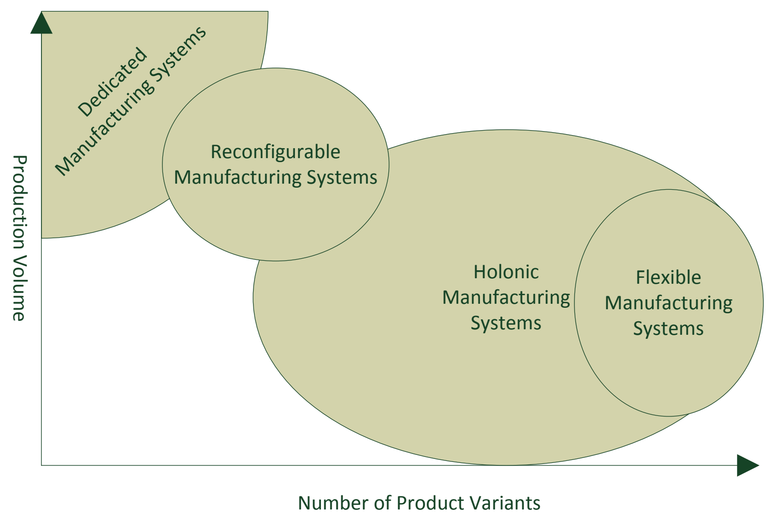

Dedicated Manufacturing Systems
Reconfigurable Manufacturing Systems
Holonic Manufacturing Systems
Flexible Manufacturing Systems
Production Volume
Number of Product Variants

Figure 1.1: Selection criteria for manufacturing systems concepts.

stepwise switch the control of manufacturing machines and other components in manufacturing sites (e.g. intra-logistic components) to a MAS based control. Furthermore, cooperating software agents are also able to plan the reconfiguration of the plant itself [12, 20].

Both the DCS as well as the MAS concept integrate a component based approach. Thus it is possible to replace or remove modules, which are not necessary or apt for a product. Whenever necessary the open control architecture also allows integrating new machines in the system. But not only the production machines have to be flexible, but also the material handling system, which is connecting the single manufacturing entities. A flexible material handling system can highly increase the productivity in case of disturbances or break-down of machine or material handling components [16, 17]. Thus there are many possibilities to design or optimize a manufacturing system, since all entities have to be taken into consideration. Figure 1.1 provides a classification of the presented manufacturing systems concepts regarding the two most influencing criteria: production volume and product variants. Those two criteria contradict each other and cannot be met at the same time. High volume production requires well optimised plants. For example the masses that have to be moved should be low to allow higher acceleration rates. Specialiced, optimised tools can fulfill such a requirement. On the other hand high number of different products (or product variants) requires adaptations of the plant. Hence, the average production rate is either reduced by reconfiguration tasks (i.e. unproductive down-times) or general purpose tools, which restrict for example the acceleration rate due to higher moved masses.

1.1.2 Manufacturing Systems Design

The products are the central elements in production system design. Starting from the product design, the production process is split into single manufacturing and assembly operations (e.g. turing, milling, glueing, moving) [21]. These tasks are then put into relation—some tasks can be performed independently from each other, others are dependent and have to be performed consecutively. Product variants also have to be considered, since the manufacturing process has to be adapted. Some tasks will have to be replaced or modified. Therefore, the overall system has to be flexible in order to handle such product variation [21].

The identified manufacturing and assembly operations form the basis for the selection of production machines and material handling components. Other criteria include the expected number of products, number of variants, or quality. To set up an efficient manufacturing system layout, some a priori estimations are possible for example with line balancing algorithms [22, 23]. These algorithms allow to calculate the average load of the included modules and to identify bottlenecks in the planned layout [6]. Furthermore the throughput of the manufacturing system can be determined. Additionally economic criteria have to be considered during the design of the manufacturing systems. The selection of the modules and the layout of the manufacturing system are not trivial tasks, since a wide range of different criteria has to be considered [24]. Better suited (e.g. higher efficiency, faster change-over times) production system structures could be provided by integrating sophisticated selection and planning algorithms in software tools (e.g. manufacturing support systems) [6, 25]. Such tools are only able to operate efficiently, and provide good results, if all existing data (explicitly including requirements) are available during the design process. However, much data do still only exist in domain-specific, segregated tools.

Several approaches try to close this gap. The Computer Integrated Manufacturing (CIM) approach aims at the integration of various tools, which cover distinct areas of the manufacturing process [26]: Production Planning and Control, Computer Aided Design, Computer Aided Planning, Computer Aided Manufacturing, Computer Aided Quality Assurance, and Maintenance. This high-level approach is a first integration step. Nevertheless, even within the various areas an integration is necessary. The integration of engineering data is a goal of many engineers and managers in the manufacturing environment. Popular concepts like the Product Lifecycle Management (PLM) and the "digital factory" approach are also based on data and tool integration [27, 28]. PLM tool chains are available from various vendors for special domains (e.g. automotive, aeronautics). However, the integration of software tools from different vendors—the usual situation in industrial environments—is hard to accomplish.

Instead of the integration of different engineering tools, the AutomationML initiative [29] is focusing on a data exchange format based on XML (Extensible Markup Language). The topology of the plant is modelled in the open data format CAEX (Computer Aided Engineering Exchange), which is standardized in IEC 62424 [30]. The automation logic of the components, described in AutomationML, is provided in the PLCopen XML exchange format. Geometric and kinematic information on the plant components are also within the scope of AutomationML. These data are specified in the COLLADA (COLLAborative Design Activity) data format. Additional aspects of plant components can be included easily. Links between the aspects are provided in the central CAEX-model, which is seen as "glue for seamless automation engineering" by the developers of AutomationML [31]. However, the loose coupling of the data renders semantic correct data exchange impossible without additional guidelines or meta-models.

1.2 Motivation

Product and production system design are separated processes, which influence each other. The missing integration of the processes leads to a lack of immediate feedback for design changes in either of the processes. Therefore important characteristics, such as lead-time, costs or producibility, can only be estimated or calculated after another design iteration. Such design iterations can be time consuming, as non-automated steps have to be repeated as well.

Even more, the implementation processes lack tool-integration. Currently production system design and production system implementation are often linked by requirements and specification documents in natural language. Natural language provides a high level of expressiveness, but confusion can be generated by different interpretations. Interfaces between the experts from different disciplines based on such documents are error prone and inefficient. The missing tool and data integration leads to rigid engineering workflows. The different disciplines work on the implementation of the manufacturing system in consecutive order in order to reduce friction losses. Additional iterations are expensive and take a long time in such an engineering workflow. Changes in the mechanical or electrical design (e.g. removing sensors) can cause high effort for the implementation of the control software.

In software engineering the concurrent development process highly increased quality of the product while reducing the development time. The enablers for this new development process are common planning, source code management systems, and software tests for validation [32, 33].

The concurrent design process for production systems design and implementation is necessary to reduce the design and implementation time (including ramp up) and reach more efficient production systems [5]. As a first step of integration of the disciplines a transition from domain specific change management to a system-assisted cross-discipline change management is necessary [34]. Changes in one domain would be immediately provided to experts from all other involved domains.

The data-integration of the "digital factory" and CIM concepts would solve these problems. However, existing implementations of these approaches only partly cover the engineering workflow. Information on the products which shall be produced (variants, amount, schedule) are provided by enterprise resource planning as well as production planning and control systems. These software tools are only loosely coupled or integrated [35]. Unidirectional information transfer prohibits timely feedback to customers or sales personnel.

Experts from multiple domains are involved in the design, implementation and commissioning of plants. An integrating workflow helps to overcome existing hinders and to improve the quality of the provided solution. The communication of experts from multiple disciplines, involved in the implementation and commissioning of plants, is facilitated. Furthermore, the inclusion of simulation in the design process increases the efficiency during the development of industrial automatic control applications:

- Saving commissioning time: The automatic control engineer is able to start the implementation and the commissioning of the control software earlier. Thus the need for expensive on-site time can be reduced.
- More reliable control application: The simulation of the plant behaviour allows earlier implementation and thorough testing of the automatic control application.
- Validation of fault recovery procedures: The simulation of the plant behaviour enables the injection of faults in the nominal behaviour. Thus the reaction of the automatic control system and its fault recovery strategy can be validated. Injuries and damages to the plant or the environment can be avoided and down times are reduced.
- Reuse of already existing models: In the development cycle of complex plants, often simulation is used to validate the mechanical design or chemical process. The integration of these models in the validation of the automatic control application will reduce the modelling effort and increase the quality of the simulation results.

Currently, tests and functional validation of plants are conducted during commissioning and ramp-up. Simulation-supported validation allows to identify problems earlier and save precious commissioning time. The integration of simulation in an model-based engineering workflow is promissing to improve the overall engineering efficiency.

1.3 Goal

The goal of this thesis is to improve the development and change of production systems. The proposed engineering workflow takes into account the multi-disciplinarity of the engineering process. Experts of all involved disciplines shall maintain a common model. This model acts as specification, which is accurately representing the current (or currently planned) automated system. Similar as in modern software engineering approaches this common model shall also be the source for code generation. Re-use of previously implemented, tested, and deployed units in a systematic, component-oriented approach improves the engineering efficiency. The direct integration of simulation in this engineering workflow facilitates the parallelization of the implementation tasks of the different disciplines. The proposed simulation framework for the design of industrial automatic control applications uses models of the controlled plants that are provided in the common engineering model. Hence, the validation by simulation is available during all phases of design, implementation and commissioning of automatic control applications. The control engineers shall be enabled to validate early the structure of the automatic control system as well as the automatic control application. To raise the trust in the results of the simulation efforts the validated control application shall be deployable without changes.

1.4 Guideline through the work

The remainder of this thesis is structured as follows.

Chapter 2 elaborates relevant concepts and approaches in modern software engineering. Languages in industrial automatic control environments and their specialities compared to general purpose programming languages are investigated. Furthermore, the application of simulation in manufacturing systems design is analysed and a detailed view on simulation techniques is presented. The deduction of 5 research questions concludes this chapter.

The proposed multi-disciplinary engineering approach is presented in Chapter 3. Automation Components are introduced as building blocks for automated systems, which can be put together hierarchically. The models, which act as the backbone of the multi-disciplinary workflow, are elaborated in detail. Furthermore, the integration of existing tools along the workflow is presented.

Chapter 4 introduces the simulation framework, which can be used for the validation of automatic control applications. Different scenarios, from pure simulation to deployment and testing are presented. A methodology for the specification of plant behaviour in the component oriented engineering workflow is investigated. The idea to use event-based automatic control environments, which are compliant to IEC 61499, is analysed. Special focus is put on the coupling of simulation and automatic control. The proposed Instrumentation- and Control Point concept as well as well-structured automatic control applications facilitate the engineering and transition from simulation to operation.

The implementation of the proposed modelling infrastructure is presented in Chapter 5. The integration of external specification tools, and the implementation of models is shown. Furthermore, implementation details on the code generation facilities for the automated creation of automatic control application and plant simulation applications by means of model transformation are provided. Also the simulation execution system, which is based on an event-based runtime environment is presented, including the integration of external simulation tools and the implementation of special simulation functionalities. The validation and evaluation of the proposed approach by the means of three test cases from different domains conclude this chapter.

Finally, Chapter 6 concludes this thesis and gives answers to the research questions on the successful evaluation of the proposed engineering and simulation approach. Future work as well as potential new applications of the proposed approach are provided.

CHAPTER 2

State of the Art

2.1 Software Design

Software has been taking over more and more functionality in industrial automation during the last decades. However, due to the close relationship to the automatic controllers, such as Programmable Logic Controller (PLC), and the controlled hardware, the methodology for the development of control applications is still at a low abstraction level. On the other hand, methodologies for pure software products (e.g. business software) have greatly improved efficiency. Reuse of previously developed functionalities and a higher software quality have been reached [36, 37].

2.1.1 Model Based Design

Models are used for a long time in most engineering disciplines (e.g. aerodynamics). They represent entities of the real world and can be used as a basis for a common design process and for validation purposes.

Also in computer science and software engineering models are used in the design process. Model Driven Software Development (MDSD) is a widely used term for the application of models in the design process of software products. As many artefacts of a software system as possible shall be generated automatically from formal models [38, 39].

Formal Model Different kinds of models are used in software projects for various purposes. Some sketches or Unified Modeling Language (UML) diagrams on paper may represent design aspects. However, not all models are suitable as basis for MDSD, due to the lack of an appropriate formal basis [39]. A formal model has a defined grammar, which includes its structure (syntax) as well as its meaning (semantic). Thus a formal model is able to describe a distinct aspect of the software in a complete and unambiguous manner. All stakeholders of a software development project shall be able to understand the model. These requirements imply that a single model does not have to and will not be able to describe the whole system. But it has to be clear, what is included in the model [39].

Artefacts The most important result of MDSD is executable software. Therefore, artefacts that are derived from the formal model include source code of a programming language (e. g., C++, Java, C#). But also files and data required at the runtime of the software can be generated, such as configuration files. The third group of artefacts support the development process itself. Software tests and documentation, for example, are part of this group [38].

Automatic Transformation The step from the formal models to executable software is automated to a large extend in MDSD [40]. The need for manual programming (i.e. transformation from specification to code) is replaced by a model transformation infrastructure. Model transformation rules describe the interdependencies of unambiguously defined entities in the formal source model and entities (e.g. code fragments) in the target environment. Hence, later changes and refinements in the model can be easily, and repeatedly transformed into executable code [38, 39].

MDSD Approaches

Model-Integrated Computing Model-Integrated Computing (MIC) is a development approach that advocates the systematic use of Domain Specific Languages (DSLs) throughout the system development process and lifecycle [41, 42, 43]. The field of application ranges from small-scale real-time embedded systems to large-scale enterprise applications. In the MIC paradigm, application developers model an integrated view of the entire application, including the interdependencies of its components (software and hardware). Models are placed in the centre for the entire life-cycle of computing systems, including specification, design, development, verification, integration, and maintenance. Using MIC technology one can capture the requirements, actual architecture, and the environment of a computing system, generally in the form of high-level models. Validation and verification of functional as well as non-functional requirements is of high importance in this approach [42]

Generative Programming Generative Programming is a software engineering paradigm that has the goal to automatically build optimized software products [39]. For supported platforms, elementary, reusable implementation components exist. They provide each specific functionalities, with a minimum level of redundancy. Furthermore, these components can be combined in many different ways to provide the specified functionality. The (static) generating entity has detailed knowledge on the pre-existing, elementary software components of a platform. Thus it is able to decide, if the requested configuration can be produced for the platform, and if so, to provide the optimized product [39]. C++-Template-Metaprogramming is implementing the Generative Programming paradigm [44, 45].

Model-Driven Architecture Model-Driven Architecture (MDA) is an open standard which is provided and maintained by the Object Management Group (OMG) [46, 47]. The main goals of MDA are interoperability, portability and reusability through architectural separation of concerns [39]. For that reason MDA introduces (at least three) different models: Computation Independent Model (CIM), Platform Independent Model (PIM), and Platform Specific Model (PSM). The CIM captures system requirements as well as the situation in which the system is expected to be used. Domain experts shall be involved in the creation of this model, which is independent from technical feasibility of a later system implementation [38, 48]. Requirements for the system are brought to a technical level in the PIM. Due to the technical (platform independent) feasibility further requirements are added to the previously specified requirements (i.e. provided by the CIM). The PIM describes the system, but does not show any details about its use of a platform [48]. The PSM captures all technical details of the system for a specific platform. The PSM is gained from the PIM by transformation. Additional information about the specific platform is added during the transformation process. The MDA is open about the source of these additional data. It might come from Platform Models (PMs) or transformation patterns [48]. The transformation process within MDA can be done manually, computer assisted or fully automatically. The main idea of MDA is to provide reusable, platform independent specifications of systems in form of models. MDA is tightly linked with other OMG standards like UML (Unified Modeling Language) or CORBA (Common Object Request Broker Architecture) [49].

Modelling

The different MDSD approaches use models with diverse scopes. These models have to represent the captured concepts unambiguously. General purpose languages are capable to do so, but a more efficient approach is to use specialized modelling languages. These modelling languages have been developed for the description of specific aspects, setting the contained information in a special context. Therefore modelling languages are able to reach a high level of expressiveness [38].

> *"We do not believe that a single modeling language is suitable for all embedded systems. Rather, embedded systems should be modeled using Domain Specific Modeling Languages (DSMLs) that are tailored to the needs of the particular domain." Karsai et al.* [42]

Each language, which can be formally described and is accessible to generators, is a modelling language [38]. Graphical languages, textual languages as well as tables are apt, as long as the syntax and semantics of the content are following rules.

Modelling languages are usually defined in meta-models.

Meta-Modelling

A meta-model has the same relation to a model as the real world to this specific model. Therefore a meta-model is a higher level model, describing another model. The meta-model includes information about the structure, relations and limitations of a specific model. The *abstract syntax* as well as the *static semantic* of a modelling language has to be defined in the meta-model [38].

Meta-modelling techniques are an important building block of MDSD. The meta-modelling infrastructure is the enabler of a formalisation of modelling. Models which are clearly defined on the basis of a detailed meta-model can be automatically interpreted and evaluated. Fields of application in MDSD include [38]:

- definition of modelling languages,
- validation of models,
- model transformation,
- code generation, and
- software tool integration.

Domain Specific Languages

Domain specific modelling is the abstraction of facts of a concrete problem space, using a modelling language apt for this specific domain [38]. A DSL is a modelling, programming or specification language which is built for and restricted to a particular problem domain [50]. This focus as well as an appropriate notation allows the efficient abstraction of domain specific concepts. *"The limited expressiveness of DSLs makes it harder to say wrong things and easier* *to see you've made an error."* [50]. The formal nature of DSLs allows domain experts as well as machines to understand the captured content of the model [38]. Thus DSLs facilitate communication between domain experts and other project members, who are involved in the system development process. However, the focus of a DSL to a certain domain or even problem class within a domain (with the limited expressiveness) does not allow to describe concepts of other domains.

Model Transformation

Model transformation is a process that creates something else (e.g. model, code, text) from a formal model [39]. Therefore model transformation is an important element in MDSD as it facilitates the linking of different models. For an automated and reproducible model transformation a formal definition of transformation rules is necessary [38]. There exist two different kinds of model transformation:

- Model-to-Code (M2C) / Model-to-Text (M2T) transformation: M2C transformation generates source code out of a formal model. Therefore it is also called code generation. This kind of transformation is necessary for the last step from models to platform dependent artefacts and is therefore a core element in all MDSD approaches.
- Model-to-Model (M2M) transformation: M2M transformations transform one or more formal source models into a target model. For the formal definition of the transformation rules meta-models of all involved models are required. The models may be based on the same meta-model or different meta-models.

2.1.2 Component-Based Development

Component-Based Development (CBD), also known as Component-Based Software Engineering (CBSE) or Software Componentry, is another branch of the software engineering discipline. CBD emphasises the separation of concerns by decomposing engineered systems into functional or logical software components with well-defined interfaces [51].

Components are independent units that are not sharing internal states or data. Only explicit communication using the interfaces to exchange messages and data is allowed. Therefore components can easily operate with other components or applications [51]. Components are system elements that provide pre-defined services. They can be deployed independently and are subject to composition by third parties [51]. Furthermore, components build entities of deployment, which cannot be deployed partially [51].

Interfaces are the only possibility of components to exchange data and messages with their environment. An interface is a set of operations or services. Clients can invoke these operations, whose semantics are specified [51]. The interface specifications act as contracts. Providers and clients both have to fulfil the interface specification to enable interoperability without knowledge of the implementation of the other component. This aspect is important especially for the reuse of components and the composition by third parties.

A e-commerce system can be used as example for such a composed system [52, 53]. Different aspects and services (e.g. order system, shopping cart, payment process) are provided by different components [53]. Different components that provide equivalent services (e.g. payment service) may be used or exchanged as long as they provice compatible interfaces [52]. They may support different payment methods (e.g. credit cards, direct bank transfer), but provide the same service to the e-commerce system that uses them.

Components and their interfaces are abstract parts for the modelling of complex systems, hiding concrete implementations. The description of these components can be separated in into component and interface models [54]. In contrast to the abstract component and the interfaces, the implementations of functionality and interfaces may use states. States are energy or information storages that resemble input and output behaviour. The states can apply constraints and interdependencies of input and output ports, dependent on previous operations and communication with the environment [54].

2.2 Languages for Industrial Automation

In each domain special methodologies exist to specify and describe the application. In industrial automation the desired behaviour was implemented with pure mechanical (e.g. cam-based timing) or electro-mechanical (e.g. relay technology) solutions for a long period. Since the invention of the PLC around 1970 [6] control software has gained its place in industrial automation. Most of today's industrial automation solutions incorporate PLCs. Other solutions build upon Programmable Automation Controllers (PACs) or Industrial PCs (IPCs) running a softPLC runtime environment.

Programming tools for PLCs are widely used and accepted in the domain of industrial automation. These engineering tools have a status that is equivalent to DSLs in the MDSD concept previously introduced in Section 2.1.1.

2.2.1 Programming Languages in IEC 61131-3

In the domain of industrial automation the prevalent languages are defined in the IEC 61131-3 standard. This standard was first adopted in 1993 [55] and representatives from all major vendors of PLCs have been involved in the standardisation process. One of the main goals of the standard was to harmonize the diverging languages and guide their further development [56].

The selection and standardisation of the 5 languages for industrial automation was a compromise. Both textual languages (Instruction List (IL) and Structured Text (ST)) as well as graphical languages (Ladder Diagram (LD), Function Block Diagram (FBD), and Sequential Function Chart (SFC)) are included in IEC 61131-3 [55].

Instruction List IL is an assembler-like programming language [6]. An executable command (*instruction*) is described within a single line [57]. An instruction consists of the following elements [55]:

- Label (optional): jump marker in order to reach the instruction
- Operator/Function: IL operator (e.g. loading or storing variable, masking, jumping) with optional modifier or function (e.g. calculations)
- Operands: constants or variables for the operator or input parameters for the function (none, one or several)
- Comment (optional)

Listing 2.1: Addition of two Integer values in IL syntax.

```
LabelAdd:  (* Label *)
LD  ValueA (* Load first value into accumulator *)
ADD ValueB (* Add second value to value in accumulator *)
ST  Result (* Store value into Variable Result *)
```

Listing 2.1 shows an IL programme for the addition of two integer values. IL can usually be directly executed by the PLC [6]. Hence, it is often used as intermediate language to execute programs written in the other languages [57].

Structured Text ST is a high-level programming language for PLCs. Its syntax is comparable to PASCAL [57]. Hence, more complex statements can be expressed efficiently (compared to IL). The same functionality (addition of two integer values) as in Listing 2.1 is provided in ST Syntax in Listing 2.2.

Listing 2.2: Addition of two Integer values in ST syntax.

```
Result := ValueA + ValueB;
```

Non-Boolean values (e.g. integer, floating point) can be handled (easier) with ST [6]. Furthermore, ST allows a clear construction of programs [57]. Hence, a higher level of reuse of functionality is possible.

Like general purpose high-level programming languages ST also provides control structures (e.g. loops and selection) [55, 57].

Programs written in ST cannot be executed directly. They have to be translated into machine code. There are little to no possibilities to influence the compiler in the translation task. Therefore, the same doubts in the efficiency of the compiled programs exist for ST like for all high-level programming languages [57]. Nevertheless, this disadvantage usually only affects special applications in industrial automation.

Ladder Diagram The graphical LD language stems from the relay technology and its wiring diagrams [56]. Thus the language is convenient for the shop floor personnel, which maintains the plant and is familiar with wiring diagrams [6]. Pure LD language allows the creation of Boolean applications and is sometimes also called Ladder Logic Diagram. Networks connect the *power rails*, on the left and right of the LD [55]. Boolean input variables (e.g. switches, relay contacts) are represented by *Contacts* [57]. Results are stored in *Coils*, which represent for example motors or solenoids which are actuated [6].

Additional functionality (e.g. timers, counters) can be integrated by using functions or even function blocks within LDs [55]. In this programming method it is hardly possible to clearly structure the control application [56]. Hence, extending or reusing existing applications is difficult, as the access to input and output variables is not limited to application parts [56].

Function Block Diagram The FBD language has its origins in the field of signal processing. Handling integer and floating point values is important in this field. In the meantime this graphical language has become a universally usable language in the domain of industrial automation [57].

A Function Block (FB) is the basic building block in the FBD [55]. It encapsulates functionality (algorithm and data) in a reusable way [56]. This methodology is an analogy to integrated circuits (ICs) in electronics design [56]. FBs provide a specified interface, where parameters and input variables are provided to and output variables set from the FB. More complex functionality is reached by combining multiple FBs to a Function Block Network (FBN). Outputs from one FB can be connected to the input of other FBs. The order of execution of the FBs is evaluated by the engineering tool. Data dependencies are used for the calculation. Feedback loops (i.e. circle dependencies) are either solved with layout information (top-left to bottom-right) or by manual intervention [57].

Sequential Function Chart The SFC language is the third programming language specified within IEC 61131-3. SFC has its roots in the Grafcet language [58]. Grafcet was the first widespread specification language to describe automation systems by several states and transitions [57]. SFC allows to partition a complex program into smaller manageable units (*steps*) and transitions, which are directly connected [55]. Each step has an associated set of actions, which are executed as long as the step is active. Each transition has a transition condition, whose fulfilment ends the execution of the previous step. Thus SFC graphically specifies the sequence of execution and even allows parallel processing of these units [57].

The actions in an SFC program can be specified in any of the 5 languages mentioned above; transitions conditions can only be specified by the other 4 languages (i.e. IL, ST, LD, and FBD) [55].

Several factors influence the selection of a specific language or a combination of languages by the users: First, the education and knowledge of the programmers and existing applications within the company. Second, the application domain and its specific requirements (e.g. sequencing). But there are also regional preferences in the choice of programming languages [57]. German programmers preferably use the textual languages IL and ST; SFC is the prevalent language in France; Asian and US users mainly use LD and SFC [57]. Also control system vendors differentiate in the support of the different languages. However, these choices are closely related to their "home" markets' preferences.

2.2.2 IEC 61499 for Distributed Control Applications

Shortly after the adoption of the first edition of the IEC 61131-3 standard, the same standardisation committee started to work on a standard for distributed control, the IEC 61499. To avoid time synchronisation issues in distributed control programs (e.g. due to cyclic execution), a more general execution paradigm, the event-driven execution, has been introduced. Hence, application parts running on different control devices can be triggered by events that are transmitted via a communication network [59]. But events may also be useful for applications which run on a single device. Especially if circle dependencies (i.e. feedback paths for data) are present in a programme, the execution order of FBs cannot be unambiguously determined by the means of IEC 61131-3 [57, 59]. Events allow the explicit specification of the control flow and the execution order in a portable and consistent way [59].

For being able to visualize and grasp the events, an adopted FBD has been introduced [57]. This is the only specified language for control applications. Functionalities are encapsulated in FBs. The FBs feature event in- and outputs additionally to the data in- and outputs (like IEC 61131-3 conform FBs). To ensure distributability global variables are prohibited in IEC 61499, as these would require to synchronise them on multiple devices [59]. Data exchange within control applications is only allowed via the specified interfaces of the FBs. This increases the reusablity of the FBs, but also brings the FB-based language closer to Object-Oriented Programming (OOP).

Functionalities provided by IEC 61499 compliant FBs can be more complex as IEC 61131-3 compliant ones. FBs can contain multiple algorithms [59]. The algorithms may be specified in any language as this would be beyond the scope of IEC 61499 [60]. However, often languages defined in IEC 61131-3—especially ST—are used [57]. The Execution Control Chart (ECC) has been introduced to specify their internal execution behaviour dependent on input events and conditions [60]. Another type of FB can contain further FBs. Thus hierarchical function aggregation is possible. The control applications are better structured and reusability is further increased.

During the control application design multiple FBs are instantiated and inputs and outputs (i.e. for both events and data) are connected. At this level there is a clear separation of interface design and the implementation of the functionality. For the user of the FBs it is important to understand the behaviour of an FB. Service sequence diagrams have been introduced to describe the behaviour of FBs at their interface. This is helpful, as it supports a clear separation of interface specification and internal behaviour implementation.

2.2.3 Evaluation

The programming languages defined in IEC 61131-3 are well accepted in the domain of automatic control. Their abstraction level is comparable to early general purpose programming languages (e.g. PASCAL). Due to this low abstraction level, modern software development and design techniques (e.g. object oriented programming) are hardly applicable. Furthermore, the architecture in IEC 61131-3 based automatic control systems is based on global variables (e.g. for I/O access). This renders re-use of parts of previously implemented automatic control application an complicated task.

The modelling language of IEC 61499 is targeting a clear design and reuse of functionalities. Such functionalities are encapsulated in FBs, the interface and expected behaviour are clearly specified. Hidden interfaces, like global variables are banned, as they hinder an easy reuse of previously defined and implemented functionalities. However, the standard imposed no restrictions regarding the languages for the implementation of the provided functionalities. So any programming language of IEC 61131-3 as well as any general purpose programming language may be used. The guiding principle of distributed control requires a possibility to specify a control flow across multiple devices. Therefore, events are introduced that allow to unambiguously specify the execution order of FBs in the automatic control application. However, event-based execution also has some disadvantages. First, event-based execution and the management functionalities, that allow online reconfiguration of the automatic control applications, introduce an overhead in the execution of the automatic control applications [61]. Second, the calculation of an overload of the execution system, which is necessary for the guarantees regarding real-time execution, require a higher effort than scan-based execution semantics [62]. To overcome these challenges, Yoong et al. propose to compile the automatic control applications, that have been specified with means of IEC 61499, to a static application without a runtime environment [61]. However, this eliminates valuable management functionalities, that can be used for online reconfiguration of automatic control applications [17, 62, 63, 64, 65], and neglects the principle of configurability (i.e. any compliant tool shall be able to configure any compliant runtime environment) [60]. Configurability is considered in the standard IEC 61499 as multi-vendor setups are expected to be common in future manufacturing systems.

2.3 Simulation in Manufacturing Systems Design

"The purpose of simulation is insight, not numbers." Banks et al. [66]

In the engineering field the term "simulation" is usually referred to the activity of modelling a real component, a system, or a specific behaviour of a system with some kind of mathematical means, along with its solution. For simulation we nowadays normally intend "computer simulation", i.e. modelling a system on a computer. Currently, computer simulation is typically used when we are not able to obtain a closed form for the solution of a mathematical model, and if real world experiments are not possible [66, 67, 68]. *"Simulations are expected to provide numerical measures of performance, such as throughput under a given set of conditions, but the major benefit of simulation comes from the insight and understanding gained regarding system operations."* [66]

Before choosing a simulation based engineering approach, the applicability of simulation, especially for economic aspects and scalability, has to be tested [66, 69]. Simulation has the potential to replace costly prototypes and thus shorten engineering cycles [27]. Iterations in the development are easier and shorter. Thus the resulting manufacturing system is potentially better.

Currently simulation is mainly used in two distinct areas of manufacturing systems engineering [70]. On one hand design and optimization tasks on the factory level are supported by simulation. On the other hand, simulation support for the engineering of closed loop control (e.g. of drives) is widely used.

2.3.1 Factory level simulation

The simulation at factory level is an important enabler for the digital factory concept. Starting from available engineering information, the effectiveness of the modelled factory is validated. By means of simulation, measures for system performance are evaluated, (e.g. [66]):

- *throughput under average and peak loads;*
- *system cycle time (how long it takes to produce one part);*
- *utilization of resources, labour, and machines;*
- *bottlenecks and choke points;*
- *queuing at work locations;*
- *queuing and delays caused by material-handling devices and systems;*
- *effectiveness of scheduling systems.*

For the abovementioned system performance measurements, only coarse models of the plant components are needed. By representing machines, work cells and material transport systems as abstract models, the simulation of big plants or factories can be performed in a reasonable time.

The pre-simulation analysis and simulation planning have to ensure that the abstract models, representing the plant components, have the same behaviour as the plant components with respect to the simulation scenario.

The simulation projects are often realized in specialized simulation environments, such as Arena [68, 71] or Plant Simulation [72]. However, many other general purpose simulation environments, especially discrete-event simulation tools (e.g. Enterprise Dynamics [73], AnyLogic [74]), are apt to investigate factory logistics and material flow.

Plant simulation methods are also an integral part of most digital factory concepts, which emerged during the last decade [75, 76]. The digital factory concept is evolving from descriptive tools, based on digital models, towards a basis for plant optimisation already during the plant design phase [75]. Digital Factory was put on the strategic research agenda of the European Factories of the Future Research Association (EFFRA) by representatives from both industry and academia [77]. The most benefit from such a concept is expected in industries with static factory layouts and high volume production, such as the automotive industry. However, the functional behaviour of the machines is usually represented by simplified models [78].

In all simulation environments Manufacturing systems have to be modelled by simulation engineers. However, most tools already offer comprehensive component libraries for material flow and logistics simulation. The selected components have to be configured according to the real plant. To gain reliable results from the simulation, the selection and parameterisation of the plant components in the simulation model is essential. Therefore, expert knowledge on the chosen simulation tool as well as on the modelled plant is required. Most of the simulation frameworks offer a graphical representation of the model and the simulation results. 2D and 3D visualisation helps to understand the results of the simulation project and present them to a larger audience. With the availability of a comprehensive simulation model, consequences of design choices can be evaluated early and regularly during the plant design process. The effects of changes in the factory layout (physical placement of machines or logical, like a changed material flow) can be evaluated without the need of moving machines, stopping the operation of the plant, or even without an existing plant.

Comprehensive product design is provided by PLM tools. Manufacturing processes, especially assembly, are tackled from the product development perspective. PLM tools also allow high level simulation of such production processes. However, product and production design are still quite distinct fields of application. Further integration, like proposed by digital production concepts [27, 28, 79], would allow the evaluation of producability. The simulation system could be used as support system in order management and customer communication. Before taking an order an estimation on lead time, production time, and costs could be given.

2.3.2 Simulation in Automatic Control Design

The requirements in the automatic control domain are constantly increasing. Energy efficiency or higher throughput for example demand a systematic design approach. The identification of the controlled system allows creating a comprehensive (mathematical) model. Based on the model of the plant, the automatic control system (e.g. controller, sensors) can be selected.

In addition to explicit mathematical solutions, simulation is a widely used tool for that task. The comprehensible presentation of simulation results is another reason for the application of simulation. Changes to parameters can be validated without the risk of damaging the plant, workers or the environment. The use of simulation allows to repeatedly performing reproducible evaluations. Thus optimization algorithms may be applied to identify the best control strategy.

A classification of 4 different simulation and operation-scenarios for validating and commissioning the automatic control of machines is presented in Figure 2.1 [80]:

- Testing,
- "Reality in the Loop",

- "Soft-Commissioning", and
- offline simulation.

All of these validation methodologies have their field of application.

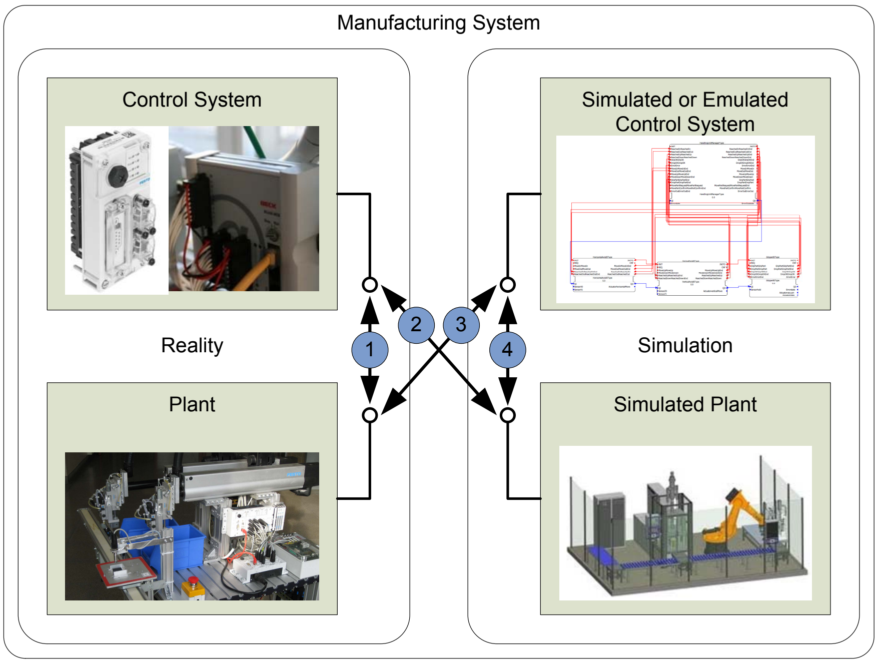


Figure 2.1: Simulation and operational scenarios [80]. 1: operation and conventional testing; 2: Soft-Commissioning and Hardware-in-the-Loop simulation; 3: Reality in the Loop; 4: offline simulation.

Testing

In this scenario the implemented control system is operating and connected to the real plant or a physical prototype. Hence, testing can only be used during or after the commissioning phase. Both the built up plant and the final automatic control system have to be available. The tests have to ensure that the specified behaviour of the automated plant and quality measures (e.g. throughput, cycle time) are fulfilled. The results either trigger further improments or allow the ramp-up of the plant. Furthermore, changes in the plant or control system at later points of the lifecycle (e.g. maintenance phase) can be assessed.

Reality in the Loop Simulation

For Reality-in-the-Loop (RIL) simulation the controlled plant has to be present. The real plant is controlled by the simulated control system. RIL simulation can be used to evaluate non-functional requirements (e.g. real-time constraints) for the control system [81]. The RIL approach is also used to optimize control systems (open and closed loop) under real-world conditions. The interface between the controlled process and the development environment (usually PC based) can be realized either by special, real-time capable hardware or with fieldbuses or industrial Ethernet connections. Real-time interfaces and prototyping hardware is provided for example by dSPACE [82] or National Instruments [83]. Inputs and outputs can be directly accessed in the according software environment. Such RIL setups are also used for the identification of plant characteristics for control design (e.g. closed loop control).

Hardware in the Loop Simulation

Prototyping and simulation coexisted for a long time isolated from each other [84]. Hardware-in-the-Loop (HIL) simulation connects these two approaches. The real controller, loaded with the developed control code, is connected to a simulation environment. The simulation environment is running a model of the controlled system and replacing the plant, interacting with the real controller. HIL simulation has its origin in the aerospace industry, where it is used since flight control systems' safety is critically affected by software [85]. But also testing the functionality of electronic control units used in vehicles is an early application of HIL simulation [85]. The need for extensive testing (e.g. aggressively driving the vehicle, driving in desserts) has been reduced. HIL simulation also has the advantage of reproducibility.

The use of HIL simulation in manufacturing systems design is motivated by several reasons:

- The need to shorten the development cycles [84] and reduce of time to market [86] is pressing for the manufacturing industry.
- To fulfil safety requirements and to prevent costly failures during the operation of the plant A thorough testing of automatic control systems, with constantly increasing complexity, is desired [84, 85].
- Increased reuse of existing components and systems is demanded due to the economic pressure and the expected reduced costs [84, 85]. The compatibility of these components has to be pre-tested to ensure a stable operation.
- The training of operators has been identified as a possible field of application [87]. Reaction in critical operating situations which could damage the machines or harm personnel can be repeatedly simulated.

The HIL simulation system has to react on events of the control system in the same way and time as the real plant. An efficient real-time computing system is needed as well as a detailed model of the plant [87]. Many experts of different domains are involved in the manufacturing systems design. Therefore the model generation, which needs good knowledge on the plant, is not an easy task in this field of application. Furthermore, the high level of details in the models does hinder the scaling to bigger systems, like plants or factories. A large amount of resources is required to execute detailed models of such large systems. Hence, real-time requirements, imposed by HIL simulation require high performance computing. That is the main hinder for HIL simulation becoming a common method in manufacturing systems design and operation. An early implementation of HIL simulation in manufacturing systems design, the Soft-Commissioning architecture, is running a commercial discrete event simulation software on general purpose PCs, which are connected to the PLC [80]. Round trip times smaller than 100 ms are reached, which is sufficient for the virtual commissioning of many components [88].

Maturana et al. [89] directly include into a PLC a special simulation device, which provides virtual I/O data via the backplane. However, such direct integration requires support from device vendors. Fieldbusses are used to provide I/O access between the controller and the controlled plant. Ferrarini et al. [90] also use the fieldbus to connect the simulated plant to the real controller. The simulator provides the virtual sensor values and actuator interfaces via Modbus/TCP. Besides the presented HIL setup, fieldbusses can also be used for coupling the real controlled process with simulated controllers (i.e. RIL simulation). Switching between simulation and operation can be done by reconfiguring the fieldbus system. However, not all fieldbus systems can be used in HIL setups without adaptations. In particular the requirement for a single master or the availability of the specifications limit the choice of fieldbus systems. Moreover, the selection of a fieldbus system also restricts the choice of the target control system, which imposes a big restriction early in the design process.

Offline Simulation

Offline simulation is the last validation methodology. Neither the control hardware with process inputs and outputs nor the controlled plant are necessary for the offline-simulation. In the industrial control systems design the simulation of the whole system, including control system and plant, modelled within a single model can be found quite seldom. Ogurek et al. [91] introduce a special block in MATLAB/Simulink [92], which is able to interpret a subset of the Structured Text language defined in IEC 61131-3 [55]. They use it to validate control programs and provide evidence for the applicability in the design and optimization of automatic control systems for process technology and municipal sewage systems [91]. However, full support for Structured Text is not possible due to restrictions from MATLAB/Simulink. Frey et al. use a Modelica based model of distributed control systems to evaluate the influence of the network topology on the temporal behaviour [93, 94]. Usual means and tools can be used for the development of the automatic control applications also for these Software-in-the-Loop (SIL) setups. A simulated controller is running the application and interacting with a simulated machine within the simulation environment [87].

Realistic Robot Simulators follow such an SIL simulation approach. Several vendors of industrial robots provide simulation environments that allow operating their products within a virtual environment. For example ABB provides the Robot Studio software tool [95]; KUKA is offering KUKA.sim for offline programming and testing [96]. The robot program can be downloaded into the virtual robot controllers as it would be done with the real ones. Thus testing can be done without endangering the working cell, workers or the robot itself.

For more general applications SoftPLCs can be used. SoftPLCs are programs for normal PC hardware that are able to execute PLC code. Combined with industrial PCs and fieldbus interfaces SoftPLCs are a recognised alternative to dedicated PLCs. For SIL simulation the plant simulation has to be coupled either via the fieldbus interface—similar to HIL simulation without the need for dedicated control hardware—or the control code has to be adapted to be connected to simulation tools, which can have influence on the execution. A drawback of using SoftPLCs for SIL simulation is that the execution semantics as well as the supported commands vary for different PLC vendors. Even more there exist incompatibilities between different products and product lines of established PLC vendors. Therefore the usage of SoftPLCs for validation purposes is only reasonable if the same SoftPLCs or compatible PLCs will be finally deployed in the plant.

Kain et al. [97] emulate the plant behaviour within an PLC or SoftPLC. The control task and the simulation task run on the same PLC using cyclic execution semantics of IEC 61131-3. The process image (global variables within the PLC) is the abstract representation of the plant (sensors and actuators). The simulation task is reacting on outputs of the control task (related to actuators) and changing inputs for the control task (setting sensor signals). For diagnostic purposes the plant simulation task can be operated in parallel to the real plant without changing sensor values. The simulation output (virtual sensor values) should match the real sensor values from the plant for normal operation [97].

Ferreira et al. [98, 99] present an object based engineering approach for numerical control systems. In their engineering environment EMBench for mechanical engineering they integrate the IEC 61499 based control engineering environment FBDK and a JAVA based graphical simulation tool. Although the control is modelled with IEC 61499 FBs the integrated, centralized simulation does not take distribution into consideration. A similar approach is presented by Rooker et al. [100], who integrate a commercial simulation environment with an IEC 61499 runtime environment to validate the movement of a component based robot.

Hybrid plant operation and simulation

Simulation and testing can be mixed for the validation of changes to existing plants or of the applicability of a specific component (e.g. cell) in an existing setup. Such a combination reveals further challenges. Gu et al. [84] identified the tight integration of real and virtual parts of the plant as an open issue. Most implementations only allow the interaction on the direct information layer, neglecting the spatial physical interaction and the physical exchange. Some approaches for the hybrid operation of the plant exist (i.e. partly simulated and partly operational), that take the physical exchange into account, exist. Harrison et al. [101] suggest continuing to simulate parts which would have been altered in the simulated part of the plant. As a consequence the operational parts of the plant are required like for the testing scenarios, whereas "only" simulation results can be obtained. In the Pabadis-Promise project buffers have been included between the simulated and the real parts of the plant [102]. Hence, real parts in the desired state enter the manufacturing system after the simulated machines. The results of such hybrid simulation scenarios reaches the reliability of testing for operational parts of the plant. However, the inclusion of buffers alters the plant setup and requires additional commissioning effort.

2.3.3 Identified Shortcomings

In current manufacturing systems design, process simulation is mostly used for validation purposes at higher levels, such as material flow simulation. At the machine level simulation is rarely used for automatic control design. An important reason is that automatic control design itself is lacking reuse [103]. The "copy and modify" methodology, i. e., taking an old control project as basis and adopting it for new needs, leads to unique, prototype-like implementations. These software implementations suffer from decreased maintainability and poor quality. Dead code, inefficient and unreliable implementations are inherent with this methodology. The same problems apply for simulation models. Hence, either unreliable simulation results have to be expected or some effort has to be put into maintenance and development of simulation models. Thus the effort for simulation based validation is too high to be thoroughly used. A suitable methodology that enables the reuse of validated components is a key requirement for an efficient use of simulation for validation purposes. The fact, that manufacturing systems are built from parts and control systems provided by multiple vendors, also hinders system validation by simulation. HIL as well as SIL simulation based on SoftPLCs are vendor specific techniques that cannot easily be applied to other parts of the manufacturing system validation [84]. Current approaches which integrate graphical simulation environments with IEC 61499 compliant runtime environments neglect the important aspect of distribution [100]. A single simulation entity is coupled with the control application.

For the reconfiguration of existing plants the hybrid operation is an interesting approach. However, the presented methods still lack maturity. One of the presented approaches only works for a specific control system [104]. Signal emulation, presented in another approach [101], requires the modification of exisiting and operational control applications. The support of control system vendors is required, which would reduce or event take away such effort [101].

2.4 Simulation Techniques

The field of simulation is very broad. Many methodologies and techniques are considered as simulation. Depending on the examined system and the simulation focus these techniques are quite distinct.

For many technical problems mathematical models can be found. These equations are either solved or estimated by computer programs. Systems of differential equations are for example commonly used for closed loop control applications. Also in mechanical engineering and the simulation of fields (e.g. electromagnetism, fluidics) differential equations build the mathematical foundation. The finite element method is a well known approach for finding approximate solutions. Hence, calculation time can be drastically reduced compared to an exact solution.

Figure 2.2 provides a classification of simulation techniques regarding three important characteristics: presence of time, basis of value, and behaviour [105].

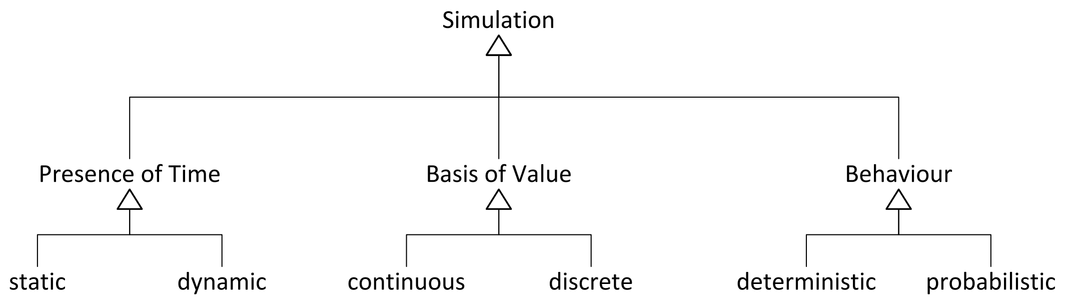


Figure 2.2: Classification of Simulation Approaches by three important properties based on [105]: Presence of Time, Basis of Value, and Behaviour.

For the simulation of control and automation systems, dynamic simulation is commonly used. The system behaviour over time is the most investigated characteristic. The simulation of chemical processes, where equilibrium states shall be reached (e.g. constant mass flows), often relies to static (steady-state) simulation approaches.

Discrete event simulation is already used in industrial practice for planning and optimization of production plants. Herein the focus of the simulation is on the interaction of the different parts within the plant and the temporal order of operations within the overall system. Instead of differential equations the models can be described by more or less simple behaviour descriptions in a discrete manner. Automatas and state machines often form the basis for such simulation approaches.

Dependent on the method of time management dynamic simulation can be classified as presented in Figure 2.3.

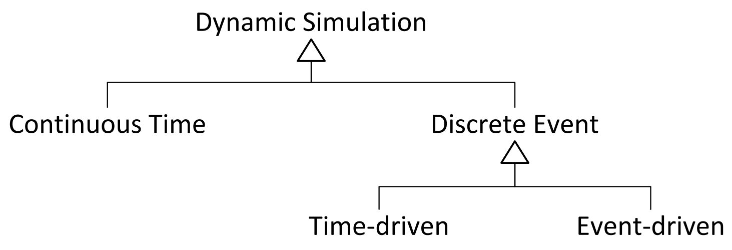


Figure 2.3: Classification of dynamic simulation by time management methods based on [105] and [106].

2.4.1 Continuous Time Simulation

The main characteristic of continuous simulation is that the models have continuously changing states. In general a number of differential equations make up these models [67]. Starting from a known initial configuration, the system states in the future are calculated and evaluated. Variables in the models assume real values and are computed for special time instants, which are separated by fixed or variable intervals. Besides the choice of the time interval also the choice of algorithms for the numerical integration is of essential importance to avoid instability and numerical errors in the results and gain reliable simulation results [67]. Based on the purpose of the simulation a trade-off has to be made between fast, but possibly inaccurate results due to simple algorithms with long intervals, and precise but computationally expensive results by utilizing sophisticated algorithms with short intervals.

2.4.2 Discrete Event Simulation

In discrete event simulation the states within the given system can only change at special points in time. For time-driven discrete event simulation these points in time can be expected periodically. Hence, difference equations can be used in the simulation models. In event driven discrete event simulation behavioural descriptions of the system are used to determine the times of the state changes (i. e., events) as well as the discrete changes of the states. Future events (i. e., events that occur at a later point in time) are determined by currently active activities. During the run of a discrete event simulation, time advances to the next future event, if all calculations and state changes have taken place at the current point in (simulation) time. The complexity of the model has a big influence on the number of events and thus on the required computational costs.

World Views

In discrete event simulation the focus can be laid on different entities in the simulation model. Three world views are prevalent in the discrete event system simulation: event scheduling, activity scanning, and process interaction [66]. Each of these world views presumes contextual information, which is not included in the simulation model.

Event Scheduling For the creation of a simulation model following the event scheduling approach, events and their effects on the system states are the main concerns. Time advancement is based on the future event list which also guarantees that events occur in the chronological order. If no more action is necessary at a point in time, the time advance algorithm advances to the point of time, specified in the first entry on the future event list. The imminent event is executed and the system state updated accordingly. If necessary, future events are generated and inserted on the future event list. Eventually the time further advances. The advancement of time is only dependent on the events in the future event list; therefore variable intervals occur [66].

Process Interaction Processes in the system are the main concern in the process interaction world view. Simulation models are based on entities or objects and their life cycle as they flow through the system. They demand resources or queue in order to wait for resources. A single entity's life cycle is a process [66]. This process consists of various events and activities. Limited resources require processes to interact (e.g. queue). Many processes may be active simultaneously in a model, causing quite complex interaction between these processes. Underlying implementations of the process interaction are usually hidden from the modellers' view; they are often based on scheduled events. Within this world view also a variable time advance, like the event scheduling world view, is used. The process interaction world view has been adopted by many simulation packages, especially in the US.

Activity Scanning With this world view the modeller concentrates on the activities of a model and conditions that allow an activity to begin [66]. Following each clock advance the conditions for each activity are checked. Activity scanning is a simple concept and leads to modular models. This improves comprehensibility and maintainability of the simulation models. However, the scanning approach results in longer simulation times than the other world views. An improved approach, called three-phase approach, is introduced, which uses features of event scheduling to allow for variable time advances also in this world view, but also keeps the main advantages. A number of activity scanning packages are popular in Europe [66].

Parallel and Distributed Discrete Event Simulation

Parallelisation of the simulation is a possibility to reduce the time demand for simulation. The concept of parallel and distributed simulation dates back to the 1970s [107]. Simulation is mainly used by engineers for analysis, for example evaluation of design changes for complex systems, as decision support system for estimating consequences of alternative options, or as basis for virtual environments, for training, evaluation, or entertainment purposes [108]. Parallel Discrete Event Simulation (PDES) and Distributed Discrete Event Simulation (DDES) can greatly improve the application of simulation in these fields by the reduction of execution time, separation of concern as well as increased fault tolerance [108, 109].

The division of the simulation model into smaller sub-models which are run concurrently allows reducing the overall simulation time. These sub-models ideally represent independent and only loosely coupled system entities. Thus also the simulation models are loosely coupled. The simulation sub-models may operate concurrently as long as the real world counterparts of the simulation sub-models operate autonomously [107]. The reduction of overall execution time is ideally proportional to the number of simulator instances, which means the execution scales well [108]. Simulation is distributed over multiple computers in both PDES and DDES. On one hand PDES is executed on tightly coupled computers, such as symmetric multiprocessing (SMP) systems and massively parallel processing (MMP) systems [105]. On the other hand DDES is executed on loosely coupled computers. Typically the frequency of interaction is smaller than 1000 per second. Failures of single simulation devices (such as failure of a processor) are tolerated in PDES or DDES as long as critical parts of the overall simulation do not fail [108].

Splitting up the simulation model imposes the need for coordination [110]. Above all, time management is necessary, ensuring that the execution of the simulation is synchronized. Time management algorithms expect that time-stamped messages or events are exchanged by the sub-models [108]. The correct execution order of the events by each sub-model, the local causality constraint, is ensured by synchronisation algorithms. If the local causality constraint is adhered by each sub-model, the results of parallel or distributed simulations are the same as of sequentially executed simulations [108]. In the late 1970s the first synchronisation algorithms were introduced [111]. These are nowadays referred to as conservative synchronisation algorithms. The sequence of execution is enforced by conservative synchronisation algorithms with additional synchronisation messages, which control the time advance.

Optimistic synchronisation algorithms, the second family of synchronisation algorithms, allow the violation of local causality. Consequences that would arise from this violation have to be inhibited by the algorithms. The Time Warp mechanism is a well-known representative of an optimistic algorithm. Whenever causality is violated, a rollback of time to a state, where causality is reinstated, is performed [112]. Historic information, that is all intermediate states and messages, has to be saved requiring additional memory [108]. Reverse computation is another technique to reinstate causality. Rollbacks are realized by performing the inverses of the individual (stored) operations [113].

2.4.3 Co-Simulation

In interdisciplinary development teams it is difficult or even impossible that experts from all domains use the same simulation tools as the engineering and analysis approaches are fundamentally different [114]. Although there exist some simulation environments that include aspects of different domains (e. g., Matlab/Simulink, Dymola/Modelica, Ptolemy) specialized tools provide higher functionality with less effort to domain experts [115]. However, the tight integration of the different disciplines is necessary for optimal system designs. The aspects from multiple domains are interdependent because of physical coupling [116]. Co-Simulation is the coupling of multiple simulation tools, each maintaining its own model, for an integrated simulation [117]. The set-up of a co-simulation environment needs expert knowledge [116]. The tool specific models have to be adjusted and unique interfaces between the involved simulation tools have to be defined and implemented [116, 117]. The synchronisation of the connected entities is a main issue [118].

The use of middleware can leverage the integration task [119, 120]. The middleware replaces direct interfaces between tools, which have to be defined and implemented for data exchange and synchronisation. The general purpose CORBA middleware[3] was used to connect a commercial SoftPLC and Matlab/Simulink. An additional tool orchestrates and synchronises the simu-

[3]Common Object Request Broker Architecture. CORBA is a communication middleware which is specified and maintained by the Object Management Group (OMG). Further information at: http://corba.org (last access: 20.09.2011)

lation tools with a project-specific message exchange protocol [119].

The IEEE standard 1516 - High Level Architecture for Modelling and Simulation (HLA) - defines the infrastructure and an interface specification that simulation tools have to follow in order to interoperate [121]. Multiple implementations of the according middleware HLA-RTI (run time infrastructure) exist, commercial as well as open source implementations. The HLA infrastructure and interface definition have proven the syntactic interoperability of commercial tools [120]. However, the models have to be adjusted to allow this interoperation [122]. Furthermore simulation experts have to ensure the semantic integration of tools, providing a common meaning of the exchanged data [120, 122].

These limitations lead to scarce application in production and process systems design, although the introduction of co-simulation has a large potential as support for concurrent engineering [123].

2.5 Research Questions

The creation as well as the reconfiguration of manufacturing systems requires expertise from multiple disciplines. Dependent on the target products the resulting manufacturing systems and plants can become complex systems, which require high engineering effort. Due to missing tool support the design and development steps of the involved disciplines are still mostly organised sequentially. Hence, the design choices of engineers later in the sequence are already restricted by previous decisions. Suboptimal solutions are the result. The automatic control engineers, which are usually the last in the design workflow, have to ensure, that the specifications are met.

Despite the high effort in control engineering, the existing tools for programming control applications are based on old software engineering methodologies. Primitive programming languages like IL and ST are widely used. These hinder the systematic reuse of existing and tested control software parts. Often system integrators use previously programmed control applications as basis for new manufacturing systems. However, this "copy and modify" approach is often limited to the same control hardware. Porting to different control systems (probably even from the same vendor) requires the programs to be written from scratch.

In software engineering new methodologies have been introduced. CBSE and MDSD have proven to increase efficiency in the field of business software development. Furthermore, the maintenance of reusable software components helps to increase the software quality and to reduce defects. From this the first research question (RQ) of this thesis arises.

RQ1: *Is a model-driven approach feasible for the design of manufacturing systems, with a special focus on control applications?*

Model-driven approaches help to reduce the design-time in software development projects. In particular the reuse of previously developed models and components as well as the parallelizing of the specification increase the efficiency. However, CBSE and MDSD cannot be directly used in industrial automation. In particular the multi-disciplinarity (e.g. interaction with hardware) brings additional requirements such as stateful components and interfaces. But also various design methodologies are used in the disciplines, which are well accepted. Hence, the second RQ emerges.

RQ2: *Is MDA apt to build the basis for an integrative engineering approach, that allows to parallelize the work of experts from multiple disciplines?*

Simulation and emulation are valuable methods to check the feasibility of proposed alternative implementations in many engineering fields. These methods have the potential to support the development of industrial control applications at an early state (i.e. without an existing implementation of the physical plant). This provides the control engineers the possibility to systematically develop the control application at an earlier point in the development cycle. Hence, the resulting control applications can have higher quality at the beginning of the commissioning phase. Nevertheless, currently available simulation frameworks (commercial as well as academic) require changes to the control system or application. Either the control algorithms need to be re-implemented on the target control system (e.g. for pure simulation approaches), or the I/O access, for example, needs to be changed for the transition from HIL setups. Changes to the automatic control applications before the deployment may introduce faults, which are not assessable with simulation and emulation. This leads to the third RQ.

RQ3: *Is it possible to validate by simulation a automatic control application and deploy it to the target control system without any change?*

Kain et al. [97] present their approach to emulate the plant behaviour on the SoftPLC. However, the implementation is based on cyclic execution of the used PLCs and therefore has some major disadvantages for simulation and emulation of large plants. First, only time-driven discrete simulation is possible. Second, if the simulation has to be distributed to multiple PLCs the synchronisation of the devices is hard to solve. Event based execution is currently emerging for automatic control systems. From this the forth RQ arises.

RQ4: *Is an event-based automatic control runtime environment apt for the execution of distributed, discrete event simulation?*

Currently simulation is already used for the validation of specific problems in various engineering disciplines for production system design. To gather reliable simulation results discipline specific simulation environments and tools are used, leading to incompatible simulation models. There exist some approaches that try to model and simulate automatic control systems in a single tool [91, 93, 94, 97]. However, there exist aspects in most simulation environments which are not well covered. A co-simulation setup that couples different simulation environments could help to reuse existing simulation models and to reduce the specification effort. However, setting up co-simulation environments usually requires expert knowledge and additional engineering effort [116]. Especially for hybrid simulation scenarios the integration of deployed automatic control systems has to be ensured. Hence, the fifth RQ emerges.

RQ5: *Is it possible to efficiently integrate other simulation tools (e.g. network simulator, continuous process simulator) into a co-simulation environment based on an event-based automatic control runtime environment?*

CHAPTER 3

Collaborative Model-based System-Design Methodology

Experts from multiple disciplines, like mechanical engineers, electrical engineers and automatic control engineers, are involved in the design of manufacturing systems. The constantly increasing demand for small lot sizes requires unique, flexible production systems [124, 125]. Thus it is hardly possible to reuse existing designs without modifications.

Furthermore the multi-disciplinary approach provides a wider field of solutions. Optimisations of the manufacturing system are therefore also possible in all involved disciplines. However, such discipline-specific optimizations might yield a suboptimal solution for the whole system. Experts in the different disciplines have unique knowledge. Also the approaches and methodologies to solve problems and to design systems differ. These differences imply that the experts use tools, for example software, which are supporting the specific approach, but may not be suitable in the other involved disciplines. Changes and decisions in one discipline have influence on the specification and implementation in the other disciplines as well (e.g. additionally required functionalities). A change of the mechanical structure for example will most probably also induce changes in the electrical and control sub-systems [3].

For the efficient use of resources during the manufacturing system design a co-ordination of the involved experts is necessary. The current practice is to sequentially develop the plant. This approach has two major drawbacks. First a globally optimal solution will hardly be reached, since all experts focus on the local optimum. Second the duration of the design process can hardly be reduced. Another important aspect is that the software development is turning into the most cost intensive discipline in manufacturing systems design [4, 126]. However, currently control software development may have limited design options due to the design decisions taken in the other disciplines at an earlier step.

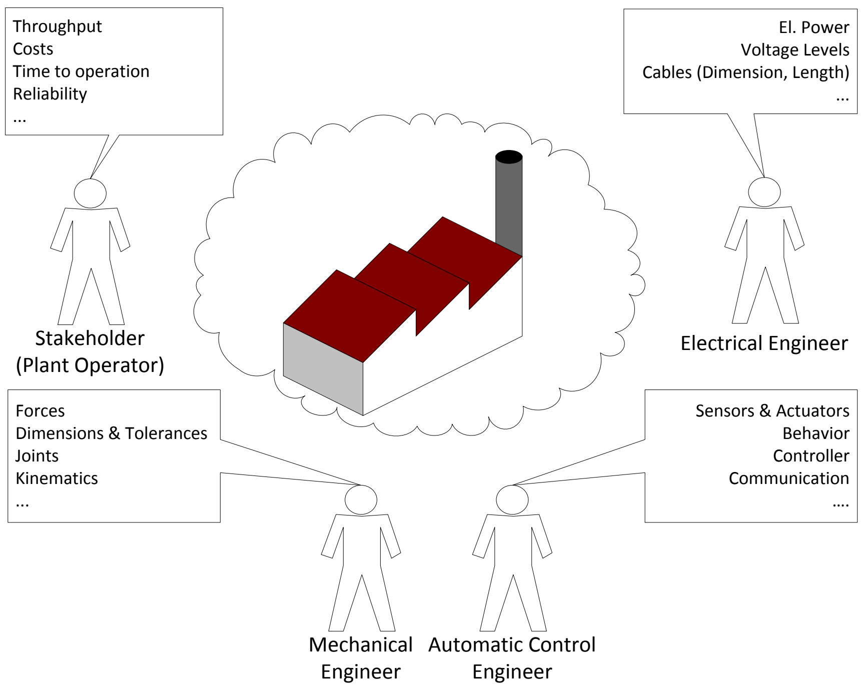


Figure 3.1: Discipline specific goals and needs in the plant design hinder the collaboration.

3.1 Envisaged Workflow for the Multi-disciplinary Engineering of Plants

An important improvement is the strengthening of the cooperation of experts from the different disciplines during the specification and design phases. With common discussions and integrative design workflows the design of the manufacturing systems will be more efficient [5]. Since the specification is done together also the implementation can be done in parallel. Thus a shortening of the time-to-production, that is the time from specification to ramping-up the plant, could be reached. Also the efficiency of the plant design might be increased.

3.1.1 Integrative Data Model

In each discipline different needs and design goals are prevalent. Figure 3.1 illustrates such discipline specific goals for the design of production plants. For fruitful discussions during the design process it is important that the needs and approaches can be clearly communicated between all involved experts. The involved designers apply different approaches and methodologies to reach their discipline specific goals. These differences lead to models that

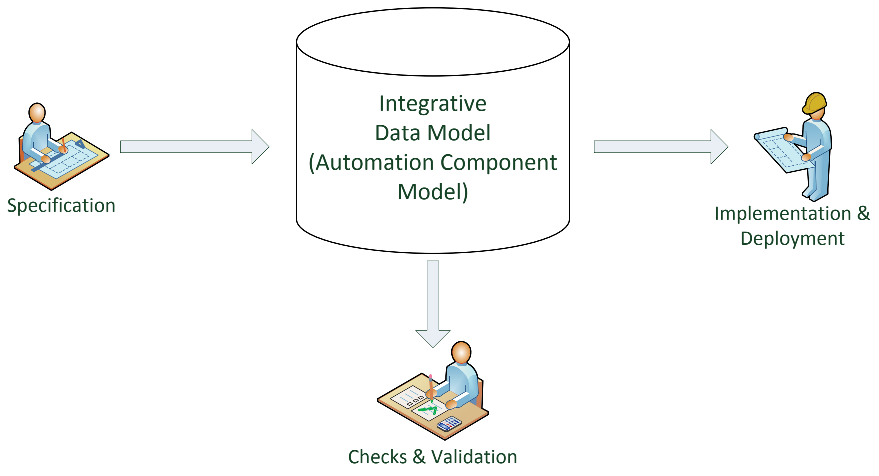


Figure 3.2: Overview on the envisaged workflow for the design and engineering of plants.

are hardly understood by experts from the other disciplines.

This challenge is tackled with an integrative data model for automation applications. This data model, the Automation Component Model (ACM), is incorporating the engineering data of all involved disciplines. The integrative data model is the central element in the envisaged design and engineering workflow (see Figure 3.2). Information provided by the various disciplines shall be linked within the data model and accessible for the other disciplines. Hence, the common data model of the designed system helps to foster the communication of the experts. However, since the model will represent and encapsulate the design data of very diverse disciplines, it will hardly be directly understood by any of the involved experts. Therefore, model transformation will be facilitated to make the stored information accessible in discipline specific form.

3.1.2 Integration of Computer Aided Engineering tools

Computer Aided Engineering (CAE) tools are well accepted throughout the various engineering disciplines. They allow the modelling with a domain specific approach and methodology. For instance mechanical structures are designed with Computer Aided Design (CAD) tools. But also engineering environments for automatic control are CAE tools. CAE tools capture discipline specific engineering data in data models. It is proposed to link the tool specific data models and the integrative ACM. Hence, the design-relevant data is captured by these tools and provided to the integrative data model. Relevant changes in the ACM have to be brought back to the discipline specific tools. Such they can be presented in all affected disciplines. The extraction

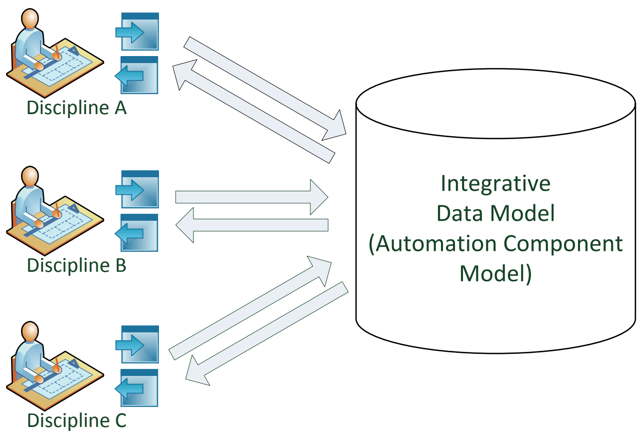


Figure 3.3: The integrative data model as well as information extraction and conversion allow the continued use of discipline specific engineering tools.

of information from the integrated data model and conversion to well-known discipline specific representations can foster the cooperation during the design phase. Furthermore, a continued use of the tools, integrating them in the proposed model based engineering approach, seems important for the acceptance of practitioners. Figure 3.3 provides an overview on the integration of discipline specific tools in the design process. The central integrative data model is exchanging the relevant engineering data with these discipline specific CAE tools.

Hence, the users of these domain specific engineering tools receive relevant information on changes in the other disciplines in their own methodology. Needs for changes in their discipline are thus easily grasped. Nevertheless, due to different levels of expressiveness of the discipline specific engineering tools not all information and changes can be visualized and checked in these tools. Hence, the modelling infrastructure has to provide a generic model viewer and ensure the consistency of the model.

3.1.3 Consistency Checks and Validation of the Model

If multiple engineering tools access the same data model at the same time, inconsistencies could occur. To prevent problems during the implementation phase, potential inconsistencies or invalid specifications shall be identified as soon as possible. For this reason discipline specific as well as discipline independent validation approaches have to be used. Basic consistency checks and appropriate reports can be directly applied to the integrative data model. Such checks could provide information on sensors or actuators that are not connected (electrically) to the control system, for example. Also the use of sensors and actuators within the modelling of the control behaviour can be analysed (e.g. unused sensors, multiply used actuators). Additionally well-established approaches for the validation should be used (e.g. simulation, verification).

These provide a basic confidence in the specification gathered within the integrative data model. But even more, the specification can be tested more precisely if for example boundary conditions, which are usually set by the user of the validation methodology, are extracted also from the integrative data model. An important goal is also to enable simulation or verification in disciplines, which currently do not use such approaches.

3.1.4 Parallel Implementation

A consistent and complete specification for all involved disciplines is provided after the application of the checks and validation methods. Thus, the disciplines can start with the implementation within the boundaries of the specification. Opposing to currently used consecutive workflows, the disciplines will be able to work on the implementation in parallel [5].

Whenever changes in common parts of the specification are needed, the affected disciplines can cooperatively find a suitable solution. As depicted on the right hand side of Figure 3.2 the integrative data model can be used as basis for the implementation and deployment. The tools for the implementation process gather their information from the data model. Depending on the discipline these tools provide (on the basis of the common specification process) for example:

- Bill of Material (BoM),
- Schematics,
- I/O-Lists, and
- Control Code (Stubs).

3.1.5 Structuring the Plant Specification

The introduction of a model based engineering approach alone is not sufficient to foster the reuse of engineering knowledge and data. The data model has to provide means to cooperatively develop a common structure of the plant, which represents needs and knowledge of the involved experts. Thus it acts as guidance for the communication and inputs. The experts have to split the designed plant into independently deployable units—this property has to be fulfilled in all involved disciplines. These units have to provide a specified interface (including all involved disciplines) and internal structure (e.g. behaviour, sub-units, electrical, mechanical design).

As with other modelling and design tasks, the choice of borders has an important role. Splitting the plant into smaller units allows thorough testing and validation. Furthermore the encapsulated functionality is graspable more easily. Interdependencies within the designed manufacturing system become visible by the introduction of borders and interfaces. This clear separation gives the chance to replace parts of the plant by equivalent components. The interfaces provide variation points and as long as the replacement units fulfil the interface contract the overall behaviour of the manufacturing system will remain the same. Section 3.2 will provide a detailed specification of the envisaged structure of reusable components for plant design.

3.1.6 Reuse of Previous Designs and Re-engineering Plants

In the software engineering domain CBD is recognized for increased reuse and better code quality (see Section 2.1.2). As soon as a set of independently deployable units is fully specified and tested, it can be used as library for future design tasks. The initial top-down-approach of splitting the functionality of the plant into smaller units can be complemented by a bottom-up-approach. Pre-existing units can be put together to provide the specified interfaces and fulfil other requirements. Hence, reusing units which have been specified and validated earlier reduces the design time and increases the system quality. Libraries containing the required engineering data could be provided by component vendors to support the design and integration process.

Due to the defined interfaces it is possible to replace one unit by another unit, which provides the same (required) interface. Thus, this approach is also applicable in the operation, maintenance, and adaptations of plants. The integrative data model can act as single source for requirements, specifications, and documentation, as long as the data is put under revision control. Influences of new, planned plant configurations on the existing plant are clearly visible. The selection of components during the reconfiguration of the plant is supported by the existing and maintained engineering data, representing the current state of the plant configuration.

3.2 Generic Automation Component

Manufacturing systems are built from engineered artefacts from different disciplines that have to be combined and whose operation has to be coordinated To increase the comprehension we split the system into independent units. These units have to be independently deployable in all involved disciplines. An important aspect of components for automated production systems is the controlled hardware. For that reason the mechanical structure of a manufacturing system provides a good guidelines for splitting the system into independent units.

These independent, automated units have interfaces (e.g. mechanical connectors, electrical connectors) and provide services. Combined and enriched with the corresponding engineering data such a unit comprises a so called Automation Component (AC), which is a building block in the proposed engineering approach. Software components have properties, which make them suitable as basis for the proposed design approach for automated systems. Independent deployment and third party composition are also important in this domain. However, software components, as they are coined by Szyperski [51], do not provide externally observable states. That means, a given component at all times provides the same output to the same given input. Hence, the same component (i.e. only a single instance) can be used multiple times in a system, if its functionality is needed. There is no need to provide multiple (software) components of the same type.

Within physical systems, like manufacturing systems, multiple instances of the same component (e.g. robots) have to be available and usable. Even more, hiding states from the component interface is also limiting. The positions of mechanical parts, which interact with the environment, are already providing state information. Furthermore, diagnostic data (e.g. error states) might be considered for passing it to other components.

Figure 3.4 depicts an example for a basic AC—a PowerCube rotating module from Schunk[4]. Engineering data from design domains, in this case mechanical, electrical and control engineering, are provided together with the physical module. This allows to provide the AC across company borders, where they can be used to build manufacturing systems. However, not all engineering data that are necessary for engineering the internals of such a PowerCube, will be provided to customers. It is sufficient to provide detailed interface and service descriptions. To protect Intellectual Property (IP) the provided engineering data can be reduced. Then the component can be used as is, but changes to the internals of the AC are restricted.

More generally, an AC is characterized by its exposed interface (elements from all involved disciplines), its services (i.e. the behaviour that is related to given inputs at its interfaces), its concrete implementation as physical module, and the corresponding engineering data,

$$AC = \{ACInterface, Services, PhysModule, EngineeringData\} . \tag{3.1}$$

As previously defined, an AC has to be a deployable unit in all involved engineering disciplines. All interfaces exposed by the different disciplines (e.g.

[4]The German company Schunk has been an industrial partner in the EU funded project MEDEIA [127]. Schunk provides modular robot-components, which can be used for the assembly of customized robot applications (e.g. for laboratory automation).

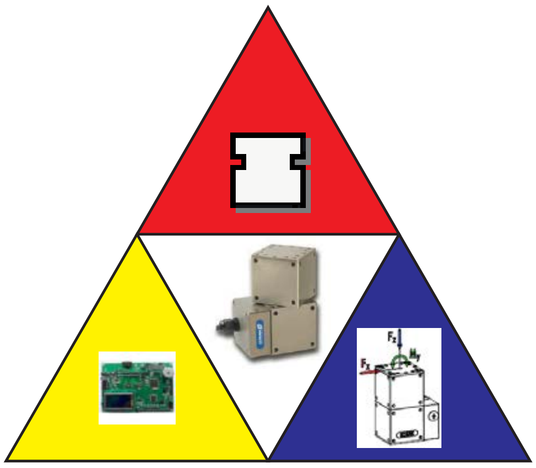


Figure 3.4: Automation Component: The rotating PowerCube module, depicted in the center, is provided together with the mechanical model (at the bottom right), the electrical model (at the bottom left), and the control model (at the top).

connectors, flanges) build up the Interfaces set,

$$Interfaces = MechIF \cup ElectrIF \cup \ldots \cup ContrIF. \tag{3.2}$$

The externally exposed interface of the AC can be reduced to a subset,

$$ACInterface \subseteq Interfaces. \tag{3.3}$$

Nevertheless, the internal (i.e. not exposed) interface elements are available for coupling the implementations of the different disciplines. To allow the reuse of ACs and the corresponding models in a collaborative manner a thorough modelling of all interfaces is important. Hidden interfaces, which are not included in the models, break the component concept, hinder reuse, and might even affect the stability of the composed system.

Services of the components are specific behaviour (i.e. output) based on the input at the component's interface. ACs provide services in each discipline. For instance the PowerCube shown in Figure 3.4 provides a service for the rotational movement, which can be seen as its main service. However, additional services are also provided like for reading and setting position data and running mode (e.g. inhibited, operable).

$$SubServices = MechServices \cup ElectrServices \cup \ldots \cup ContrServices. \tag{3.4}$$

The previous set comprises of all discipline specific services. However, new additional services will be provided by the ACs. For example the rotational movement will not be triggered by applying voltage to the motor, but a target speed or position is set via the control interface. Different sub-services,

$$SubService \in SubServices, \tag{3.5}$$

are used as building blocks for these new services.

$$Service = \{SubService_1, SubService_2, ...SubService_n\}. \tag{3.6}$$

Thus, an AC is more than just the collection of discipline specific data, as new services are the result of collaboration during engineering and design. The set of services provided at an ACs interface,

$$Services = \{Service_1, Service_2, ...Service_n\}, \tag{3.7}$$

might be, but does not have to be, a subset of SubServices (as defined in 3.4).

The last aspect, which completes the AC (see Eqn. 3.1), is its physical part.

$$PhysModule = \{MechModule, ElectrModule, ContrApp, ...\} \tag{3.8}$$

An ACs Physical Module comprises of all discipline specific implementations (i.e. modules). These modules are the providers of the discipline specific interfaces and sub-services. Also computational hardware, which is needed for the execution of the control applications, must not be neglected.

ACs are designed to provide specific services with their physical module and interface. By combining multiple ACs new tasks can be fulfilled.

For illustration purposes a 4 degrees-of-freedom (DOF) robot on the basis of three separate ACs is presented in Figure 3.5. Two rotating PowerCubes (each 1 DOF) and one tilting and rotating PowerCube (2 DOF) comprise the robot.

Equivalent to basic ACs also this newly created entity is characterized by its interface, services, physical module, and engineering data. Hence, this hierarchically aggregated entity is called composite AC.

$$AC_{cmp} = \left\{Interface_{cmp}, Services_{cmp}, PhysModule_{cmp}, EngineeringData_{cmp}\right\}. \tag{3.9}$$

The approach is comparable to modules or sub-assemblies in the production of goods. The aggregation shall be done only following functional criteria. Hence, reusable components are reached, whose functionality can be clearly specified and provided by a self-contained component. Based on the desired (i.e. specified) functionality of a composite AC existing, or at least specified, ACs are selected and put together in a bottom-up approach.

$$AC_{composite} = \left\{AC_{sub_1}, AC_{sub_2}, \dots AC_{sub_n}\right\} \tag{3.10}$$

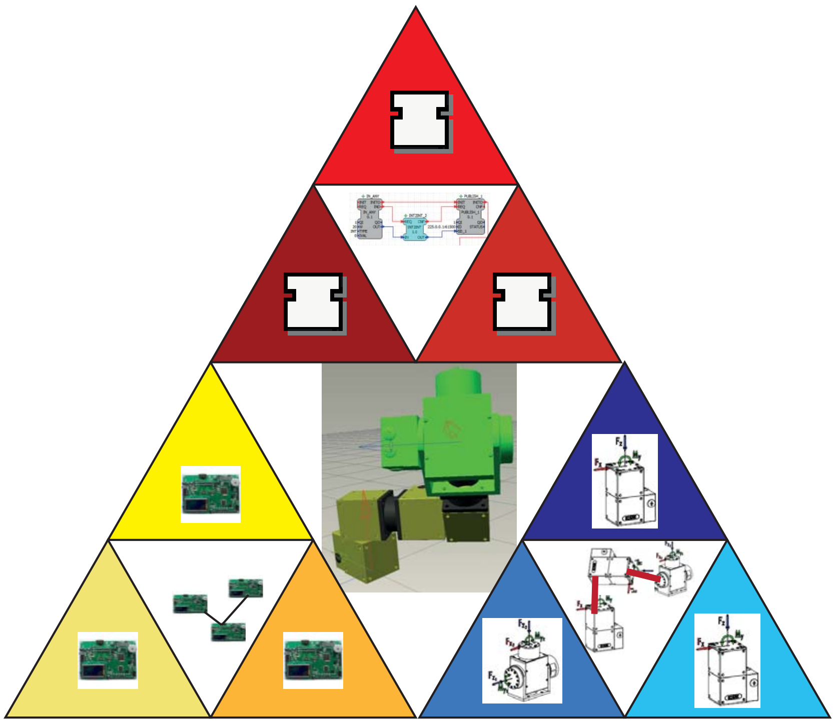


Figure 3.5: Composite Automation Component: 4 degrees-of-freedom robot hierarchically built up by aggregation of 3 ACs. The electrical, mechanical, and control parts of the composite AC emerge from linking the sub-ACs.

For the aggregation itself, engineers can concentrate on the interfaces and services, which are provided by the sub-components, to realize the specified behaviour of the AC.

The interface of the newly created composite AC is based on the interfaces of the contained ACs. Since parts of the sub-ACs interfaces are used to connect them and build the composite-AC only a subset is exposed by the composite AC.

$$ACInterface_{cmp} \subseteq ACInterface_{sub_1} \cup ACInterface_{sub_3} \cup ACInterface_{sub_3}. \tag{3.11}$$

Furthermore, the same statements, which apply for the basic ACs (i.e. 3.2 and 3.3) also apply for composite ACs.

Also the services provided by a composite AC are composed from services provided by its contained ACs, e.g. movements of the PowerCubes. However, composite ACs need to provide tasks for the coordination of the services of the sub-ACs. Coordinated movement of the axes of the robot (i.e. composite AC) is such a new service, which is using the movement services of the contained axes (i.e. contained ACs). Such management or coordination services have to be added in the composite ACs. These services would contradict the premise, that a composite AC's services are just a combination of its sub-ACs services. Two possibilities have been identified to reach a consistent modelling approach. First, additional modelling elements for such services could be introduced specifically for composite ACs. Second, the coordination services could be encapsulated in a new, pure functional AC, which is added as sub-AC in the composite AC. Pure functional ACs are components without a hardware interface. They just provide logic functionality to coordinate services of other ACs at the same hierarchical level (e.g. coordinated movement of PowerCubes).

The second approach can build on meta-models which are specified for basic ACs. Hence, it is prefered for the implementation of the meta-model for composite ACs.

For the creation of composite ACs from other ACs the nature of the used components (i.e. basic or composite AC) shall not matter. Both, basic and composite ACs are characterized by their respective interface, services, physical module and corresponding engineering data (see 3.1 and 3.9), which follow the same scheme. Only the interface and the services, that are provided at the interface of an AC, are of importance for the aggregation. Hence, the number of aggregation levels (i.e. composite ACs containing composite ACs) is not limited in the proposed engineering approach. Furthermore, the focus on the interface and the provided services allow replacing an AC by another AC as long as the (used) interface-elements and services are also provided by the replacement AC.

3.3 Proposed Engineering Data Models

An AC is composed of the actual implementation and all related engineering data. Services and interfaces of the AC are provided by the implementation, which is an outcome of the specification process.

For an increased reusability the MDA paradigm showed its potential in the software engineering domain (also see Section 2.1.1). The separation of concerns in the MDA approach also leads to multiple separate models: PIM, PM, and PSM.

In the PIM the functionality is described and specified without influence of and dependency on the targeted execution system. Hence, in pure software engineering projects the PIM if free of any hardware specific aspects.

For the automation domain the term platform has to be carefully specified. Automated processes and manufacturing systems design are based on mechanical and electrical systems that manipulate the real world in order to produce goods. Hence, the automated equipment is part of the functional design of manufacturing systems. Engineering data related to the design of the automated equipment is included in the PIM. To address the changed scope, the PIM

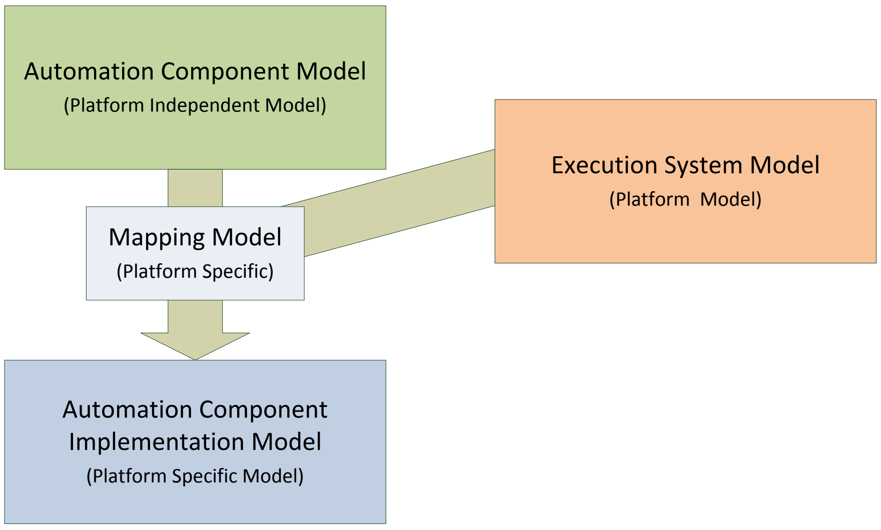

Automation Component Model
(Platform Independent Model)
Execution System Model
(Platform Model)
Mapping Model
(Platform Specific)
Automation Component Implementation Model
(Platform Specific Model)

Figure 3.6: Models for the Model Driven Automation Systems Design. Three main models and their dependencies in the modelling workflow are provided: a) Automation Component Model (ACM), b) Execution System Model (ESM), and c) Automation Component Implementation Model (ACIM). Each of the models covers different engineering aspects following the MDA approach.

is named ACM within the proposed approach (see Figure 3.6).

The execution system, which finally runs the automation application, shall be replaceable. Thus, it shall not have influence on the functionality and may be considered late in the engineering process. Therefore, the engineering data related to the execution system shall be included in the PM during the automation systems design.

Following the related MDA approach the combination of PIM and platform relevant information (e.g. PM) leads to the PSM. The resulting model—the Automation Component Implementation Model (ACIM)—is therefore the closest representation of the deployable AC. This model acts as the source model for code generation tasks. The influence of the execution system and the functionality of the AC are available within a single model for the first time in the modelling process. Hence the validation and verification of the manufacturing system as a whole is best done starting from the ACIM.

In the following sub-sections, the introduced models, their contained submodels and data will be elaborated.

3.3.1 Platform Independent Aspects

The ACM is the core model in the proposed model driven design approach for automation systems. Basically it incorporates the functionality of an AC and sets requirements which have to be fulfilled by the platform. The functionality

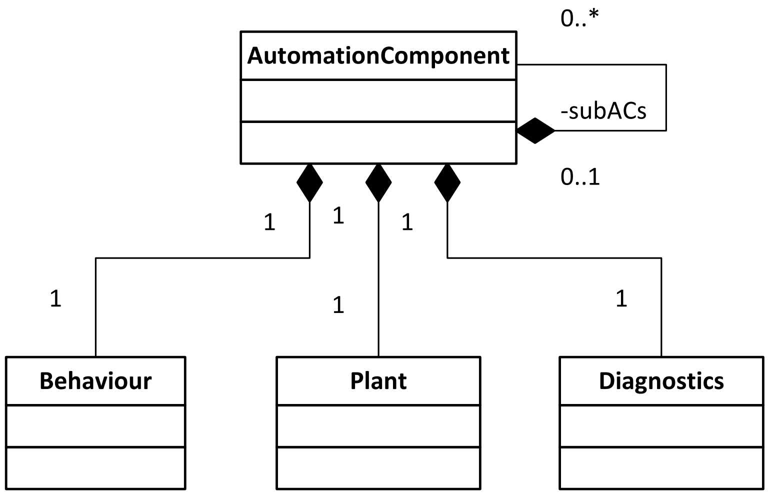


Figure 3.7: Separation of concerns in the Automation Component Model. The UML Class Diagram shows the relations of the Automation Component Model and its sub-models. Behaviour Model, Plant Model, and Diagnostic Model are included for distinct, complementing engineering aspects. For better readability the term "Model" is neglected in the specification of the Automation Component Model.

of an AC is established by quite distinct characteristics, including the automated hardware. To accommodate the different nature of the related engineering data, the ACM is split into three sub-models (see Figure 3.7): Behaviour Model (BehM), Plant Model (PlaM), and Diagnostics Model (DiagM). However, the sub-models have to be coupled closely on well-defined interfaces in order to enable an efficient engineering workflow.

Plant Model

The PlaM is responsible for hardware related engineering data. As shown in Figure 3.8 the plant behaviour, interfaces, the physical design (i.e. mechanical and electrical designs), and the aggregation into more complex composite ACs are of relevance in the PlaM.

The physical part of an AC includes mainly mechanical and electrical aspects. To facilitate the integration of ACs in manufacturing systems the explicit modelling of physical interfaces is of high importance. For the specification of the AC itself, component-internal interfaces between the disciplines are of utmost relevance. The interface between the hardware specified in the PlaM and the logical aspects of the AC, which are encapsulated in the BehM includes abstract representations of sensors and actuators. Hence, the actual interface of sensors and actuators is hidden from the specification of the component behaviour, which can thus be really platform independent.

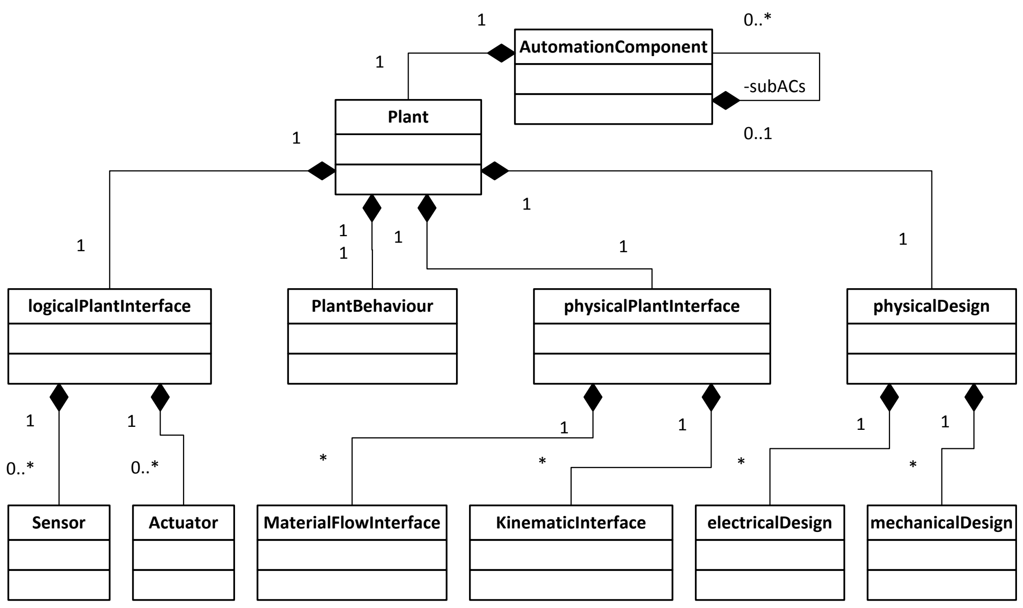


Figure 3.8: The plant model is representing the automated parts of the ACs. The specification of interfaces and the behaviour as well as the mechanical and electrical design are covered.

For the aggregation of ACs (i.e. building composite ACs) only the externally provided interfaces of the sub-ACs are relevant. Within the plant specification especially the mechanical and electrical interfaces are important. These are a subset of the AC's interface as defined in Eqn. 3.3.

During the aggregation also physical services of the ACs (e.g. material handling, kinematic movements) have to be connected. The connections specify the use of physical services at the level of the composite AC dependent on the mechanical composition. The material flow and kinematic chains have been identified as relevant for the domain of manufacturing and robotics. Therefore, the according interfaces are explicitly provided in Figure 3.8. Further interfaces and services can be integrated later, as long as a separation in behaviour and interface is possible at the AC level.

The possibility for a separation into and linking of mechanical designs of ACs and sub-ACs is also important. This allows to create mechanical models of the designed manufacturing systems from multiple ACs, as well as feeding back changes to the affected ACs. Since CAD is well used, many different tools and data formats exist. Hence, the use of an already existing CAD data format is promissing. Two data formats, which are mainly used for data exchange, fulfill the given requirements, and are suitable for the integration into the PlaM

- COLLADA is an open industry standard [30]. It supports to store both exact geometrical information and tessellated boundaries for the visualisation. Data are stored in XML files. Referencing other files allows to aggregate larger components [30]. The AutomationML initiative [29] is using COLLADA and is actively improving this data format for the application in the automation domain.

- JT (Jupiter Tesselation) has been developed for the automation domain [128]. The geometrical data are stored as binary data, additional data and aggregation information is stored in XML files. JT is directly supported by Siemens PLM software products, but also software from other vendors in the automation domain (e.g. Dassault Systems) is supporting this data format [128].

Besides the interface specifications for the physical as well as logical interfaces and the geometric data, also the behaviour of the physical parts with respect to changes at the interfaces is important. Automation engineers who implement the control behaviour of the ACs need to know the expected reactions of the component with respect to changes at the *logicalPlantInterface*. For instance sensor values (e.g. angular position of the PowerCube unit) are dependent on the state of the AC. The description of the functional behaviour is necessary for that purpose. *Service Sequence Diagrams* and *Activity Diagrams* that are part of the UML-family [129] are apt. Both diagram types also allow introducing non-functional aspects, like timing, in the behaviour description.

Behaviour Model

The BehM is the most important model for the automatic control engineer. The overall behaviour at the component interface, the specification of the logical component interface (including I/Os), and the internal (controlled) component behaviour comprise the BehM, as shown in Figure 3.9. Following the MDA approach, taking into consideration multiple distinct execution platforms (e.g. scan based, event based), a neutral behaviour description model is desired. Requirements for the specification of the behaviour have been gathered in the MEDEIA project in an extensive requirements engineering phase in four automation domains [130]: manufacturing systems, modular robotics, energy production, and packaging industry. The industrial partners provided valuable information on each domain, which built the basis for the specification of the behaviour model. Hence, during the specification of the BehM the balance between expressiveness on the one hand and support for reliable code generation for various execution platforms was important.

Specification of internal component behaviour: Timed state diagrams are well suited for the specification of the desired behaviour of an AC. On one hand, the envisaged modelling approach presented in Section 3.1 shall support

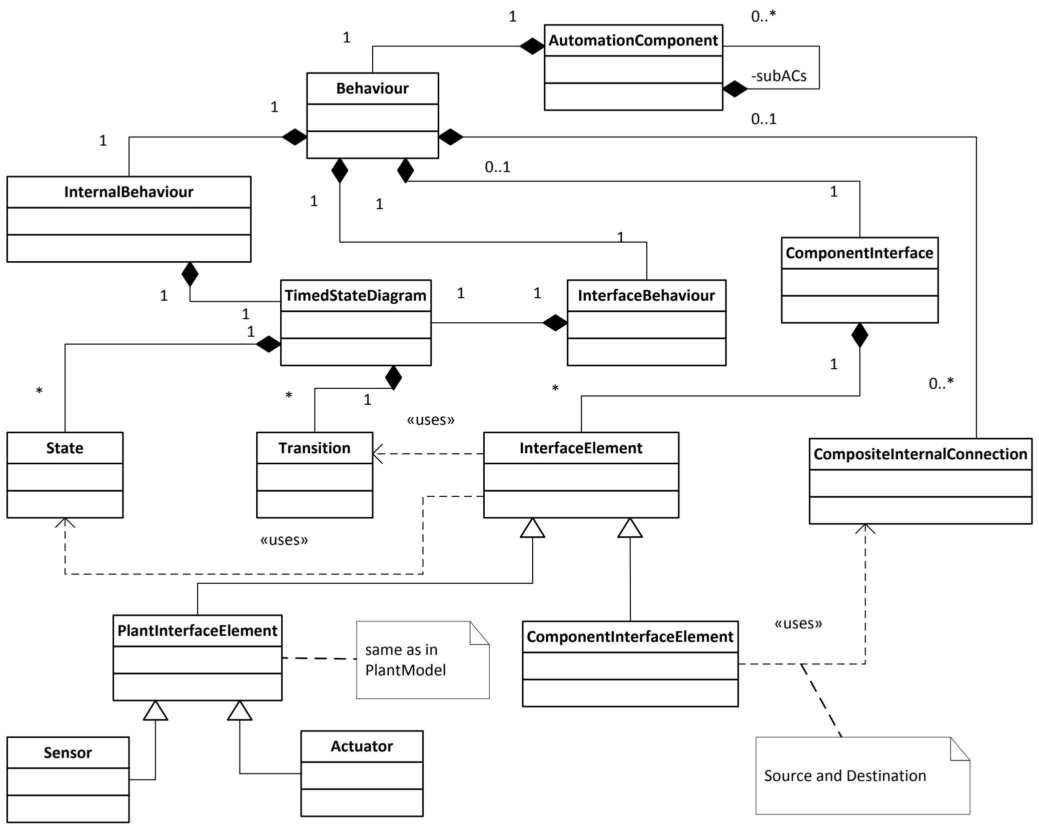


Figure 3.9: Overview on the main classes of the AC Behaviour Model. This meta-model allows to describe the functional behaviour of ACs.

multiple distinct tools. On the other hand, verification tools like UPPAAL [131] are based on flat automaton models. Hence, the flat structure of the timed state diagrams is a compromise in conciseness and expressiveness with respect to the envisaged engineering workflow. Furthermore, hierarchical structuring and concurrency which have been introduced with hierarchical state charts [132] are covered by the aggregation within composite ACs.

Within the states interface elements can be altered. That means, actuators in the controlled plant are used (via*PlantInterfaceElement*), services from sub-ACs are requested (via *ComponentInterfaceElement*), or service responses are communicated to higher-level ACs (via *ComponentInterfaceElement*). Reading access to all interface elements is provided both in states and transitions. Hence, transistion conditions can also use statuses of lower levels (i.e. aggregated ACs and plant) as well as requests from higher levels. Furthermore, the algorithms executed directly after entry into the states can use all available statuses and requests.

This way specified internal component behaviour provides information on how the plant and sub-ACs are used in order to provide the desired functionalities (e.g. services).

Specification of external component behaviour at the interface: To support the use and reuse of ACs in the engineering process it is necessary to know, how to use these ACs. The specification of the external component behaviour that is included in the ACM is important to grasp the functionality of the ACs.

The interface behaviour specification is independent from the implementation. That means, it must not have any influence on the specified interface behaviour whether the AC is a basic AC, which is directly interacting with the plant, or a composite AC, which is using sub-ACs to fulfil its functionality. In a top down approach the external component behaviour specification can act as a guide for the implementation of the AC's internal behaviour.

The external component behaviour specification is a contract of provided functionality and services. Opposed to pure software components, ACs may have stateful behaviour at their interfaces. Only the *ComponentInterfaceElements* of the AC itself are available for the specification. Hence, the interface behaviour specification has a smaller (or equal) number of states and transitions as the specification of the internal behaviour. Therefore, it is feasible to use the same specification method, the timed state diagram.

However, it is crucial that the promised interface behaviour (i.e. external component behaviour) is compatible with the implementation. Verification methods, like model checking, may be used to ensure this compatibility [133].

Specification of the Interface: The specification of the interface in the (logical) behaviour model comprises of *InterfaceElements* of two kinds: *ComponentInterfaceElement* and *PlantInterfaceElement*.

First the *ComponentInterfaceElements* build up the *ControlInterface* of the AC. The *ControlInterface* is a sub-set of an AC's interface. Together with other interfaces (e.g. the mechanical and electrical interfaces) it comprises the ACInterface (see 3.2 and 3.3). Higher level composite ACs use the *ControlInterface* to control and coordinate the sub-ACs. Electrical and mechanical interfaces are specified in an AC's plant model.

Second the interface specification also includes *InterfaceElements* of controlled entities. *ComponentInterfaceElements* of sub-ACs allow the coordination of provided services. Also the sensors and actuators of the controlled plant are represented as *PlantInterfaceElements* in the internal interface. Hence, these *InterfaceElements* can be used within the specification of the internal behaviour.

3.3.2 Platform Aspects

In an MDA approach aspects regarding the execution platform are gathered in the Platform Model. Many of the currently used engineering approaches for automated control development and deployment provide MDA-like mechanisms. Labelling I/Os and using the label instead of the address in application design is a first step of abstraction provided by engineering tools. The mapping of I/Os, which are connected to the control system via a fieldbus, into the process image also separates engineering from implementation. IEC 61499, which allows using multiple devices to execute a control application, has even more properties of an MDA approach [134]. The control application is developed independently from the execution devices in the *Application Model*, which is thus comparable to the PIM. The *System Model*, equivalent to the PM, is describing a particular hardware configuration (e.g. control devices, execution resources, network segments, network links) [60].

However, the models in the aforementioned engineering approaches are of limited expressiveness. Usually the programming tools and execution devices are provided by the same vendor. Therefore, explicit platform models can be replaced by implicit ones, which are inaccessibly embedded in the engineering tools.

To overcome these limitation a more comprehensive model is proposed. The proposed modelling and engineering approach supports a distribution of the specified AC behaviour, which is hierarchically aggragated up to the manufacturing system level, onto multiple control devices. Also different control paradigms (e.g. scan based, event based) shall be supported within the same project. This requires to explicitly model the execution system in the envisaged engineering approach.

Execution System Model

The Execution System Model (ESM) shall provide all relevant platform data for heterogeneous control systems. Some of the currently available description languages (e.g. architecture description languages, hardware description languages) do not provide means to all relevant aspects of heterogeneous execution systems. The neglected aspects are implicitly available in their field of application (e.g. homogeneous hardware platform, single fieldbus system). The specialized configuration and modeling tools are therefore not dependent on this information. However, explicit modelling of all relevant aspects is demanded to ensure openness and the possiblility to integrate different hardware platforms, communication systems, or the like. Amoung the description languages those with a generic and extensible model (e.g. Field Device Configuration Markup Language (FDCML), CAEX) allow the integration of additional modeling information. FDCML already allows to describe most aspects of automated control systems. It is an implementation and extension of the ISO 15745 reference model [135]. The standardized base model and the expressiveness of FDCML provide a good basis for the definition of the ESM.

Figure 3.10 provides an overview on the entities which comprise the ESM.

The main elements within the ESM are (networked) control devices and interconnecting networks. An execution system consisting of a single control device does not necessarily also contain a *Network* element. However, even backplanes in control devices are treated as networks, as they are based on the same mechanisms. Some device vendors also allow extending the backplane to decentralized I/O devices (e.g. Beckhoff's EtherCAT). Information on the used fieldbuses is not only needed for referencing them in the device specifications, but most fieldbus systems also need to be configured before use (e.g. setting IDs). Each of the network protocols needs specific parameters.

In order to improve the readability and to simplify the model transformation process separate classes for the network protocols have been included in the ESM (see Figure 3.10). This imposes the need to extend the ESM if new network protocols shall be supported. Such extensions do not influence existing specifications, as they are included as additional modelling elements.

The other main model element in the ESM is the *Device*. At least one device is needed in the execution system to being able to operate the plant. A device comprises of 4 elements:

- The *Description* entity is used to specify and identify a piece of control hardware. This is necessary if multiple devices are used in a control application and supports the maintenance of the plant.
- In FDCML the *Function* entity captures network-independent information and services. The function of a pressure sensor for example is to provide 0..10 V at its output, representing pressures from 0..10 bar.
- The *DeviceManager* contains all information regarding the configuration of the device.
- All hardware Interfaces of the device are encapsulated by the *Structure* entity.

Network interfaces (e.g. network interface cards), which can be used to interact with other (control) devices are represented by *CommunicationPort* elements are managed by the *DeviceManager* as all other hardware elements. *IO-Port* elements represent terminals, where sensors and actuators are directly connected to a device. Furthermore, a device can provide extension slots (*Slot* entity). Additional modules can be inserted in the slot and provide additional functionality (e.g. I/O modules) to the control device. For the specification of such modules *Module* elements are used. These use the same specification means as devices. Hence, the *Module* class is a specialisation of the *Device* class.

Besides the existence of compatible *CommunicationPort* elements two devices also need to support the same protocols in order to enable the exchange of information. From the electric specification for example DeviceNet, CANopen and other network systems based on Controller Area Network (CAN) are

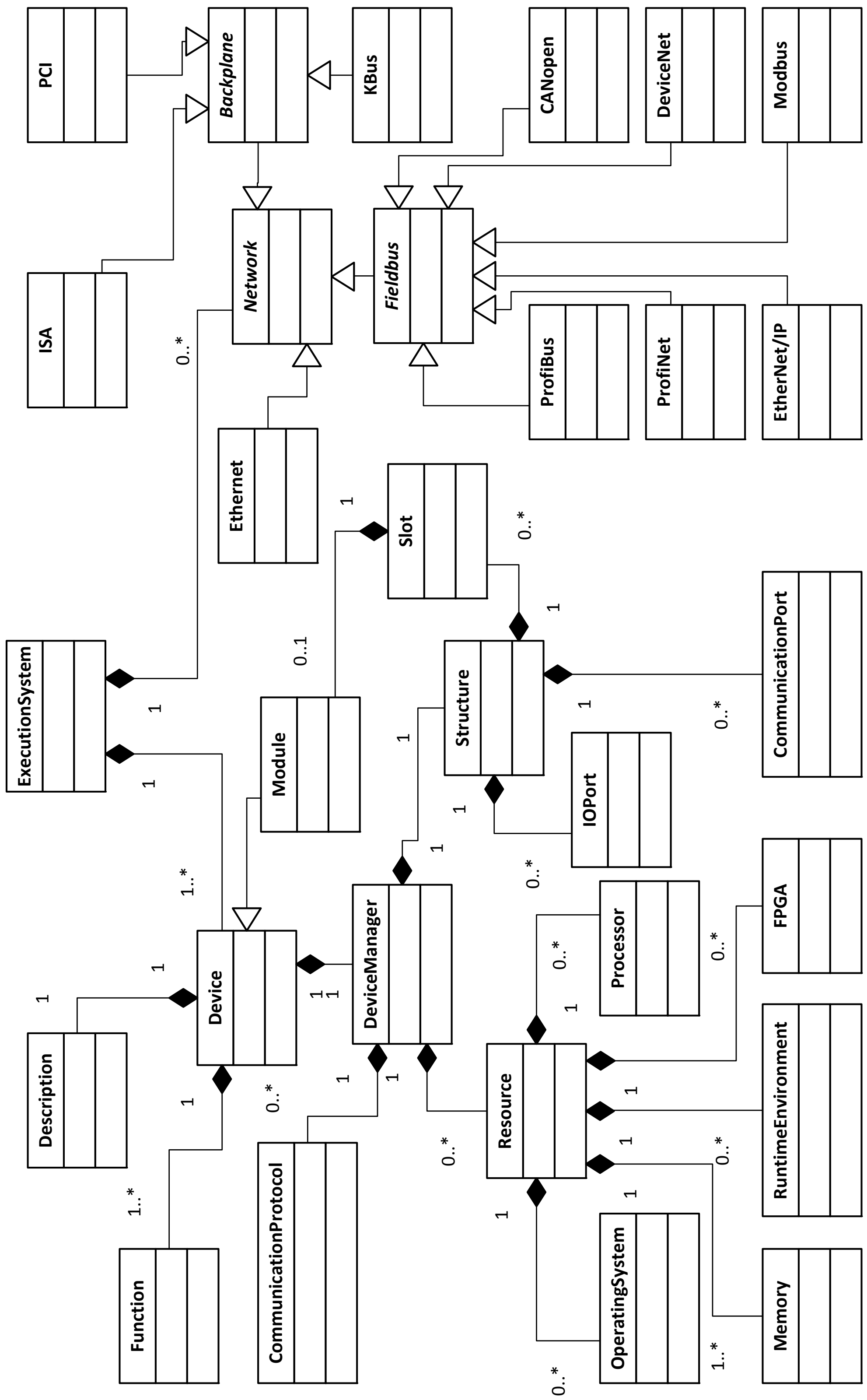


Figure 3.10: Execution System Model for the specification of all platform related information.

compatible. Nevertheless, differences in the exchanged data (e.g. encoding) and organisation of the network (e.g. IDs, Network Master) prohibit their integration. Therefore, supported protocols are important for the deployment of control applications and the control system integration. The according information is contained by the *DeviceManager* in form of *CommunicationProtocol* elements.

For the deployment and the execution of control applications other, more internal characteristics of the control devices are important. The *DeviceManager* provides *Resource* elements to specify relevant data. *OperatingSystem*, *Processor*, *FPGA*, and *Memory* are of high importance if the result of the code generation process shall be directly executable on the target device (e.g, general purpose programming language C). In cases where a *RuntimeEnvironment* is used, the hardware related elements allow estimating the load and memory usage on the target system. However, if a runtime environment is used to execute the control application, a more detailed description is needed to allow efficient control code generation. Operators and functionalities which are provided by the respective runtime environment can increase efficiency (e.g. reduced size of the programme, reduced number of CPU cycles) if they are appropriately used. Hence, for both general purpose frameworks (e.g. Java, .NET) or frameworks specialised for industrial automation (e.g. 4DIAC runtime environment, PLCs) additional specifications (e.g. version, available libraries) are necessary. Control functionalities supported by the runtime environment (e.g. encapsulated in libraries) facilitate an efficient generation of the executable control applications. Furthermore, supported execution semantics (event-based or scan-based) have to be considered during the code generation process. It is possible to mimic event-based semantics with scan-based systems, and vice versa [136, 137].

3.3.3 Platform Specific Aspects

In MDA approaches the creation of the PSM is the last step before the generation of executable code. The PIM and the PM are combined and may need to be enriched with additional information (e.g. library elements, or functions that are maintained outside of the modelling infrastructure) to tailor the PSM. Thus a specific functionality is provided by a clearly specified execution platform.

Mapping Model

The Mapping Model (MM) is provided as auxiliary model to specify necessary mappings. It links the platform independent ACM with the chosen platform provided by the ESM. For the full picture on the automated system it is not sufficient to distribute the functionality (i.e. ACs) to the appropriate devices. This is a common task in MDA approaches, however, in the design of automated

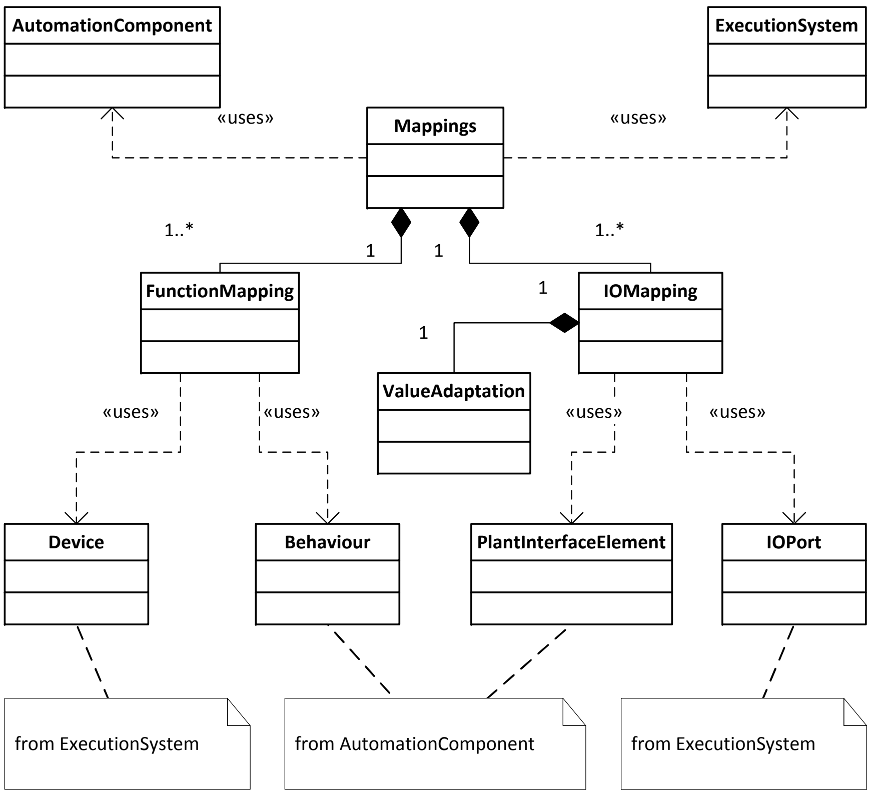


Figure 3.11: The platform specific Mapping Model links the platform independent Automation Component Model with the Execution System Model.

systems also the process interfaces towards the plant are part of the design. Therefore, the generic I/Os from the device independent BehM need to be mapped to real I/Os defined in the ESM. For the I/O mapping it is important to match the expected range of values in the logical system with the actual values in the physical system (e.g. current, voltage) and vice versa. Hence, the MM provides the *ValueAdoption* class to specify the necessary adjustments and calculations to link I/Os with their representations in the logical system. A full overview on the MM with links to the other involved models is provided in Figure 3.11.

Automation Component Implementation Model

The ACIM represents the PSM within the MDA approach. All aspects regarding the implementation of an AC are specified in the previously elaborated models. Hence, the ACIM is incorporating all these models. By model weaving [138] the information gathered within ACM, ESM, and MM is linked (see Figure 3.12) and the ACIM is created. The ACIM provides all information that is needed for the creation and implementation of the AC. Both physical pro-

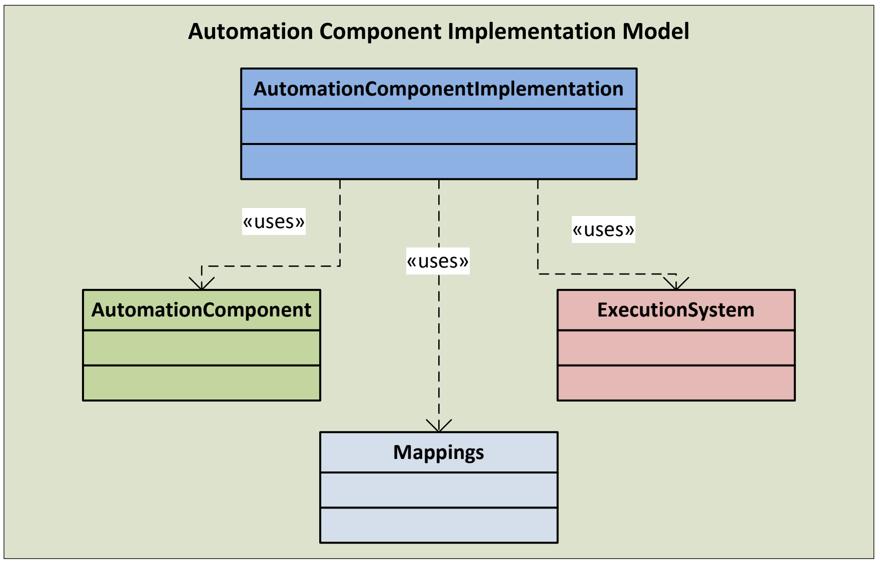

Automation Component Implementation Model
AutomationComponentImplementation
«uses»
«uses»
«uses»
AutomationComponent
ExecutionSystem
Mappings

Figure 3.12: The platform specific Automation Component Implementation Model is composed of the platform independent Automation Component Model, the platform describing Execution System Model, and the linking Mapping Model by model weaving.

perties (i.e. mechanical and electrical setup) as well as logical properties (i.e. specified control behaviour) are provided. Hence, this PSM is the basis for the implementation; also the implementation of the control software starts from the ACIM. The provided specification allows to automatically generate code or at least to generate a program skeleton with appropriate stubs.

The ACIM encapsulates all information regarding a specific implementation, both platform independent model and platform model are included and used. Hence, it is the model which is provided together with the implemented AC to customers. Thus the all the engineering data which has been specified and is necessary for the integration of an AC into bigger plants is available to the system integrators.

3.4 Domain Specific Engineering Tool Integration

The previous sections provide a good overview on the potential workflow and a detailed insight in the central modelling infrastrucure of the workflow. However, the arrows in the envisaged engineering workflow (see Figure 3.2 and Figure 3.3) hide the complexity of linking various models and tasks (from specification to code generation). In MDA approaches model transformation is used for this purpose [138].

Domain experts shall be enabled to continue using their discipline specific tools. First, they are able to efficiently use these tools. Second, already existing specifications and designs can be further used.

In order to make the information, which is gathered by these engineering tools, accessible for other disciplines in the ACM Model-to-Model (M2M) transformation is applied. On the other end of the workflow either code (e.g. executable control application) or models (e.g. verification model, simulation model) are generated. For these tasks Model-to-Code (M2C) or M2M transformation are used respectively. For the automatic transformation the data have to be represented in a well defined structure. Hence, meta-models of source and target models are mandatory.

For the central engineering repository the meta-models of the *Automation Component Model* are elaborated in Section 3.3. The provision of appropriate meta-models for all involved tools throughout the engineering workflow is required. Furthermore, semantic information on the relation of entities in the according meta-models is needed. This can be either encapsulated in transformation rules—if semantics in the models are fixed—or requested as additional user input during the transformation process [138].

3.4.1 Information Extraction

Reliable extraction of information from existing engineering data is crucial for the acceptance of the proposed collaborative engineering approach. Due to the huge variety of software tools a comprehensive integration of all these tools is not possible. Various restrictions apply. In some commercial tools models and data are exposed only to a small extent. Nevertheless, it is feasible to extract at least basic information, which is sufficient for the collaboration of different disciplines' experts. Dependent on the openness of the domain specific *source* tools, three integration approaches have been identified: direct tool integration, external transformation (vendor-specific), and external transformation (via vendor-neutral intermediate format). Barth et al. recently investigated on the openness of engineering tools [139].

Direct Tool Integration: Tight Integration of Discipline Specific Tool and Information Extraction

The direct integration of the information extraction process into a discipline specific tool is the optimal solution. First, the users do not have to leave the well known engineering environment. The interpretation of the data and information extraction can be triggered by just another entry in the menu. Second, the chance for a semantically correct interpretation of the gathered information is high. Tool developers, who would be responsible for the integration of the information extraction process, have the best knowledge about the tool internal data structures and their meaning. Nevertheless, restrictions for the design process itself might be necessary to allow a semantic correct, automatic transformation process. Such restrictions could be design guidelines, rules, or profiles (e.g. profiles in UML modelling).

Direct tool integration is highly dependent on the tool vendors/providers. Some commercial tools offer application programming interface (API) access or some plug-in infractructure. Thus, external tools can be seamlessly integrated. For open source projects the integration is feasible, if access to the used models and the meta-models is provided. Figure 3.13 provides a schematic of the direct information exchange between the discipline specific engineering tool and the ACM.

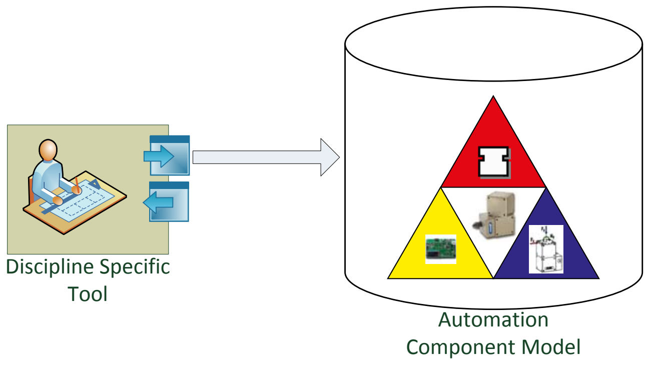


Figure 3.13: Tight integration of information extraction process in discipline specific tools.

External Transformation (Vendor-specific): Information Extraction from Proprietary Data Format

The second possible workflow to extract engineering information from discipline specific tools is based on proprietary data files. An additional tool can be implemented for the information extraction process and included in the workflow (as depicted in Figure 3.14).

The external transformation tools need an appropriate meta-model of the included information. Such meta-models are usually not exposed. Hence, only a limited set of information can be extracted without cooperating tool vendors. Furthermore, changes to the specifications of tool or vendor specific data formats are usually also not publicly announced. For that reason, external model transformation tools have to be closely bound to a specific version of the discipline specific tool.

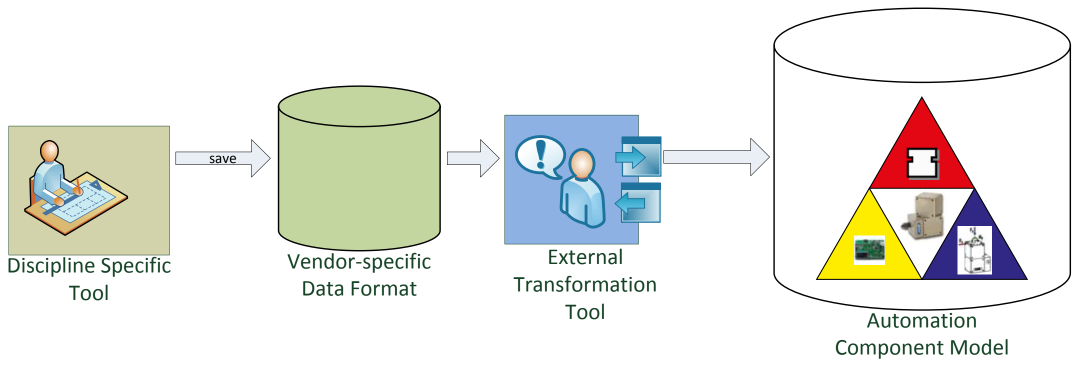

save
Discipline Specific Tool
Vendor-specific Data Format
External Transformation Tool
Automation Component Model

Figure 3.14: Information extraction from a proprietary, tool specific data format.

External Transformation (Vendor-neutral): Data Exchange via Vendor-neutral Intermediate Format

In many disciplines vendor-neutral data formats exist. Their main purpose is to enable the exchange of engineering data between similar tools. However, these intermediate data formats limit the expressiveness compared to direct data access within tools. The engineering workflow requires an additional step. First, the export to the vendor-neutral format has to be triggered. The data transformation in this step is usually provided by the tool vendors. Second, an external transformation and data extraction tool has to be used to bring the available information into the central engineering repository. The proposed information extraction workflow including a vendor-neutral intermediate data format is presented in Figure 3.15.

Although information might get lost through the two separate transformation processes, the vendor-neutral data format provides a fixed interface. Only a single set of transformation rules is necessary from this intermediate format (e.g. PLCopenXML, AutomationML, COLLADA) to the ACM. Furthermore, discipline specific tools often offer facilities for the export and import of intermediate formats. For instance PLCopenXML is supported by various PLC vendors [140]. In this case the first transformation process (from vendor-specific to vendor-neutral format) is implemented and maintained by the tool vendors. Hence, extracting information from such an vendor-neutral data format helps to integrate a larger number of discipline specific tools at once.

Summary

Currently many manual translation steps are needed in the design of automated plants. For instance printed documentation has to be interpreted and models and implementations are created by experts. To overcome such manual processes model transformation is a promising approach. However, expli-

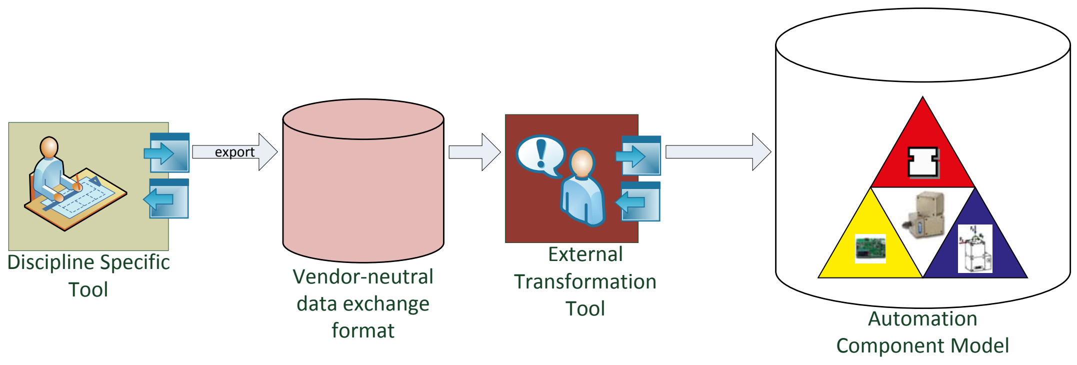

export
Discipline Specific Tool
Vendor-neutral data exchange format
External Transformation Tool
Automation Component Model

Figure 3.15: Information extraction workflow including a vendor-neutral data exchange format.

cit knowledge on the structure of the discipline specific engineering models is needed. For semantically correct interpretation of the engineering data and extraction of information, engineering tools have to strictly adhere to the meta-models. But this alone might not be enough. For example UML is a powerful and rich language. Its syntax is specified [129]. However, its semantic expressiveness needs to be limited (e.g. by a profile) in order to allow an automatic interpretation and transformation. Similar restrictions in the use of modelling elements may also be necessary in other discipline specific tools.

Three approaches for the information extraction process have been presented. Dependent on the openness of the discipline specific engineering tools the appropriate approach shall be chosen. The best integration of existing tools in a multidisciplinary engineering approach can be reached by directly integrating the information extraction and model-transformation into these existing tools. However, as cooperation of tool vendors is necessary to reach this integration level, further approaches that base on proprietary and vendor-neutral data formats have been presented.

3.4.2 Model and Code Generation

Model and code generation is the opposite workflow as compared to information extraction. The specified ACM acts as the source model. This provides a stable and well defined basis for the transformations. All entities in the source model are well known. Starting from the ACM several targets are proposed:

- discipline specific engineering tools,
- implementation specific documentation (e.g. BoM, schematics, I/O-lists),
- verification model,

- simulation model and code, and
- structured, executable code (or code stubs).

Dependent on the nature of the target, either M2M or M2C and M2T transformation is applied to the ACM.

Discipline Specific Engineering Tools

For a fruitful collaboration of experts from different disciplines at least collaboratively developed structures of the automated system have to be shared. Initial models or alterations shall not be brought into discipline specific models manually, as this is current practice [30] and bears the risk to introduce errors in the models. Hence, the discipline specific models have to receive this information as input form the central ACM. Dependent on the openness of the discipline specific tools, the transformation process from ACM to these tools is the inverse process of the previously described information extraction process. Only data in the ACM that can be represented in the discipline specific tool are used in the transformation process.

Semantic information is needed for an automatic information exchange between tools and the ACM. Transformation rules, based on entities of the source and target meta-models, statically encapsulate semantic information. To enable automatically providing discipline specific models from the ACM the ACM would need to accommodate any information provided by any involved discipline. Engineering data, which cannot be automatically interpreted as information at the time of transformation can be included in the model as uninterpreted legacy data (cf. Drath and Barth [141]) or stay with the tool. Such legacy data may be parsed and interpreted at a later moment. Furthermore, they can be included in the transformation from the ACM to the discipline specific tool, as long as the related data is unchanged. However, changes in other engineering disciplines may result in the need for changes in a discipline specific tool as well. These needed changes cannot be automatically determined with missing semantic information. Hence, inconsistent engineering models might occur. Ontology merging and mapping, which deal with similar problems, are currently investigated in the computer science domain [142, 143, 144, 145, 146]. Even semi-automatic approaches for mapping and merging are complex and challenging [147, 148].

Documentation

An important requirement for engineering processes is an accurate documentation, which has to be provided. Following the proposed workflow, the ACM acts as the central engineering data repository. Information which is necessary for the generation of documentation is collected. The most recent engineering information is the basis for such documents. Hence, it is easily possible to regularly retrieve I/O-lists, BoM for control systems, and the like. Changes, which might occur during commissioning of the automated system shall be brought back to the ACM. It is desired that also this step happens automatically via model transformation mechanisms. However, such documents are usually provided on paper. This renderes the automatic correction of the model to represent the actual implementation difficult to impossible. Nevertheless, DSLs which are tailored for the input of corrections by commissioning staff could be feasible.

Code Generation

A benefit for using a model driven engineering approach is the possibility to automatically generate code for different platforms. Verification tools and simulation environments are just additional platforms. To facilitate the generation process, meta-models and transformation rules have to be provided for each (hardware) platform. Changes in the ACM are reflected automatically to all target platforms. Hence, the typically error prone manual synchronisation becomes obsolete.

Control Code (Stubs) Retrieving control code for the automated plant is one of the core features of this proposed model driven engineering approach. Unfortunately each control device vendor (or even more, each device family of a vendor) is expecting different syntax of the applications, even if the internationally accepted standard IEC 61131-3 is used [149]. PLCopen provides a specification for a vendor neutral exchange of control applications for IEC 61131-3 control devices [140]. Control appications are not limited to automated control related languages defined in IEC 61131-3 (see Section 2.2.1) or IEC 61499 specific Function Blocks (see Section 2.2.2). General purpose languages (e.g. C, C++, Java) or specialized hardware definition languages (e.g. VHDL, VERILOG) are also potential target languages. Within a single automation application different hardware and thus different control software can be used. Hence, mapping information in the ACM as well as the device description provided in ACM's Execution System Model shall be used to provide control software for the selected devices.

The expressiveness of each of the target specific languages and the source model ACM is different. Therefore the transformation rules of the M2C transformations have to be implemented accordingly and only use relevant data (see Section 5.2). Hence, for optimization purposes manual changes in the generated code may be necessary. Furthermore, it may not be possible to provide transformation rules that provide correct code for any possible target language. In such cases only code stubs (i.e. a template for the final implementation) and the according documentation (e.g. behaviour specification) will be generated. Compared to completely manual implementation this approach provides better guidance. Relevant documentation and specification is served where it is needed.

Verification Model Verification is valuable for the validation of software [133]. Formal models (e.g. state diagram, Petri Net) are required as input. Based on these models, properties of the described software component can be evaluated. Deductive verification is usually performed by experts. Based on the specification verification properties are formulated, that can be checked efficiently by theorem provers [150]. The task of finding such properties can hardly be automated. Model checking on the other hand is stronlgy based on software tools. All possible system states as well as the transitions between the states are found and evaluated. However, with increasing numbers of states and transitions the computational effort for model checking drastically increases. This effect is called state-space explosion [151]. In the reachability graph possible deadlocks and livelocks can be detected. Also the conditions for such unwanted behaviour is provided to the user. Hence, the user can change the software to avoid them. Some tools (e.g. UPPAAL [131], Kronos [152]) also allow to add timing information. Verification on timed state diagrams also allows to validate timeliness, which is important in control systems design. However, usually only a limited number of data types (i.e. Boolean, Integer) is supported in order to reduce the complexity of the verification tasks (for instance state-space explosion).

Based on the specification in the ACM a timed model of the internal behaviour of the AC can be generated, which is compatible with UPPAAL. Hence, it can be verified that the specified internal behaviour of an Automation Component and its external behaviour at the interface are compatible.

Simulation Model/Code Simulation is recognized as valuable method for the validation of technical systems. However, in the validation of automatic control system implementation simulation is used only for selected evaluations (see also Section 2.3). One of the main hinders is the fact that currently simulation models have to be separately created. This additional effort is avoided, if other validation methods (like testing) can be applied, even if they do not give full coverage. Simulation could provide insight on the behaviour of the overall system, including hardware elements, even in harmful or dangerous situations.

Sufficient information for the generation of simulation models of the plant behaviour is available in the ACM and ESM. The information is captured during the collaborative engineering phase. An extensive concept for a better integration of simulation in the implementation phase of industrial, automated control systems is provided in Chapter 4.

Simulation models and executable code are generated from the central engineering models for the proposed simulation framework. However, also the generation of (partly incomplete) simulation models for additional simulation tools and frameworks (e.g. network simulation) is possible. Adding further modelling elements and semantic links would allow to integrate also other simulation models (e.g. based on differential equations).

3.4.3 Discipline Specific Engineering Tools in the Collaborative Engineering Approach

Engineers in the different disciplines use domain specific tools in their work (e.g. E-CAD, CAD, PLC programming environment). Hence, currently engineering data and information is distributed accross all involved disciplines. Integrating diverse, existing and used tools in the collaborative engineering workflow is a major step towards the acceptance of the proposed workflow. Domain experts can continue using of well known tools.

On the one hand, engineering information shall be extracted from existing data kept by engineering tools. For an automatic processing of the data, knowledge on the syntax but also on the semantics is neccessary. Hence, metamodels and transformation rules which incorporate semantic knowledge form the basis for the information extraction. Restrictions for different disciplines may need to be applied. UML for instance offers profiles that allow to prohibit or limit the usage of certain modelling elements [129]. Such restrictions act as guidelines in the specification process, limiting the freedom of the engineer, but enabing automatic interpretation and information extraction.

On the other hand, discipline specific tools shall also receive data. Involved domains shall be kept synchronised via the central engineering repository ACM. Hence, additonal transformation rules are required for the reverse path. But also verification and simulation tools are discipline specific tools which access engineering data of the ACM. Finally, programming environments, such as PLC programming and confirguration tools, process automatically generated code that is provided by code generators on the basis of engineering data.

The integration of discipline specific tools, their models, and metamodels is crucial for a multi-disciplinary model-based engineering approach. Specifications, which are needed for the implementation, are available consistently and early in the development cycle.

3.5 Summary

Increasing the collaboration of experts from multiple disciplines is a must to speed up the engineering process. The proposed model-based engineering approach is able to play a central role in such multi-disciplinary environments.

Feedback and discussions early in the specification process for manufacturing systems help not only to speed up the specification and implementation phases but also to increase overall efficiency. To ensure a smooth transition from currently used engineering workflows, the use of discipline specific engineering tools shall be continued. Information and engineering data, which is needed by multiple disciplines, is gathered in the central repository ACM.

A major aspect of the proposed approach is to split the functionality of a plant into smaller, graspable units—Automation Components. These components have to exist as self-contained units in all involved disciplines. Hence, the interfaces between the disciplines can be defined early in the specification phase. Another advantage of modular components is the increased reusability. In a hierarchical aggregation approach more complex ACs can be built from other ACs. The according models resemble this structure. Hence, the ACMs of the contained ACs are included in the ACMs of the composite AC.

The engineering data is automatically extracted from discipline specific engineering tools with model-transformation facilities. Syntactic information has to be provided according to meta-models for each source data model. Semantic relationships are to be encapsulated within the transformation rules. Three different approaches for the information extraction have been presented: direct integration of discipline specific tools, information extraction from vendor-specific data format, and information extraction from vendor-neutral data format. The application of these approaches is mainly dependent on the discipline specific engineering tools which shall be integrated in the engineering workflow.

The main target discipline with respect to the currently defined meta-models and transformation rules is automatic control. Automatic control engineers are currently involved in the engineering process quite late. Hence, the boundary conditions for automatic control are mostly set by other disciplines. Furthermore, an increasing number of functionalities in manufacturing systems is provided by software. Nevertheless, such software is often designed and developed in a way that does not allow efficient reuse. Hence, costs for automatic control engineering are steadily increasing.

The proposed model-based approach, including the workflow and meta-models, fosters the reuse of functionalties (including hardware and software). Despite the strong focus on automatic control, the approach is not limited to generation of artifacts for automatic control (e.g. automatic control applications, simulation models, verification models). Since information and data is collected in a multi-disciplinary way, data can be semantically linked and provided to all involved engineering disciplines. For instance, BoM and I/O-lists can be directly extracted from the currently proposed meta-models.

CHAPTER 4

A New Distributed Simulation Framework for Automatic Control Implementation and Validation

In the currently prevalent sequential workflows, the implementation of automatic control applications is the last step before the ramp up of plants. Incomplete automatic control applications are delivered, tested at the plant, and improved shortly before the deployment of the plant or even on site [80, 153]. The quality of the automatic control software is often poor because of such last minute changes.

Plant simulation enables the development and validation of automatic control applications even without the presence of the physical plant. Different scenarios, from full testing/operation to full simulation, are presented. It is proposed to integrate the modelling of the plant behaviour and the interfaces in the model-based development workflow presented in the previous chapter. This helps to greatly reduce the effort for modelling as well as for the generation and maintenance of the simulation models. Design information from multiple disciplines can be used to automatically generate simulation models to a large extent (see previous Section 3.4.2).

A further novelty is the use of an event-based automatic control environment (compliant to IEC 61499) for the behaviour simulation and coupling of the simulation to the automatic control application. In order to get readable applications, which can be transfered easily from simulation to operation, structuring guidelines are presented. These guidelines can be applied both for the manual implementation of automatic control applications and for the automatic generation in the model-based environment.

4.1 Validation of Automatic Control Through Simulation

Multiple methods can be applied to ensure the functionality of a system. Clarke et al. [133] identify simulation, verification, and testing as complementary validation approaches for software. As elaborated in Section 2.3 simulation is currently only used for specific applications in automation control design and implementation. However, reliable validation results by simulation and testing can only be reached, if the whole manufacturing system (including plant and control hardware) is equal to the later deployed system, or at least reasonably close. Automatic control is bridging the software domain with the real world (e.g. mechanical systems, electrical systems). Hence, the system boundaries in simulation approaches for automatic control implementation have to be set carefully.

Inputs and outputs of the control hardware resemble the process interfaces and are the junction of the controlled plant (hardware) and the automatic control application (software). For that reason they are a suitable and well recognizable border. Various simulation setups based on this definition of the system border are presented in Figure 2.1 and elaborated in Section 2.3.2. HIL-simulation approaches are apt for the analysis of automatic control. However, changes in the automatic control application are currently required to be represented at the interfaces (i.e. connected to simulation or plant). Changes to the finally deployed automatic control application are therefore necessary. During this process errors may occur, which cause faults in the controlled plant.

For the use of simulation as validation approach, changes to the control application itself shall not be necessary. If the need for changes cannot be avoided, such changes have to be easily grasped by the engineers who perform the simulation and who deploy the application during the commissioning phase.

4.1.1 Scenarios

Currently the most used and trusted validation scenario in automated control implementation is testing at the plant. Factory Acceptance Tests and Site Acceptance Tests are common. However, these tests are only available late in the development cycle. Identifying and fixing faults at that stage is delaying the ramp up and for that reason higher costs are imposed than earlier in the development cycle. The final control system as well as the final plant setup is required for tests. Figure 4.1 shows the components and their connections required for testing of control applications.

Restrictions to the applicability of this scenario may apply. Especially handling of fault conditions of the plant or robustness of the automated plant for breakdown of control devices need to be validated. If people or the en-

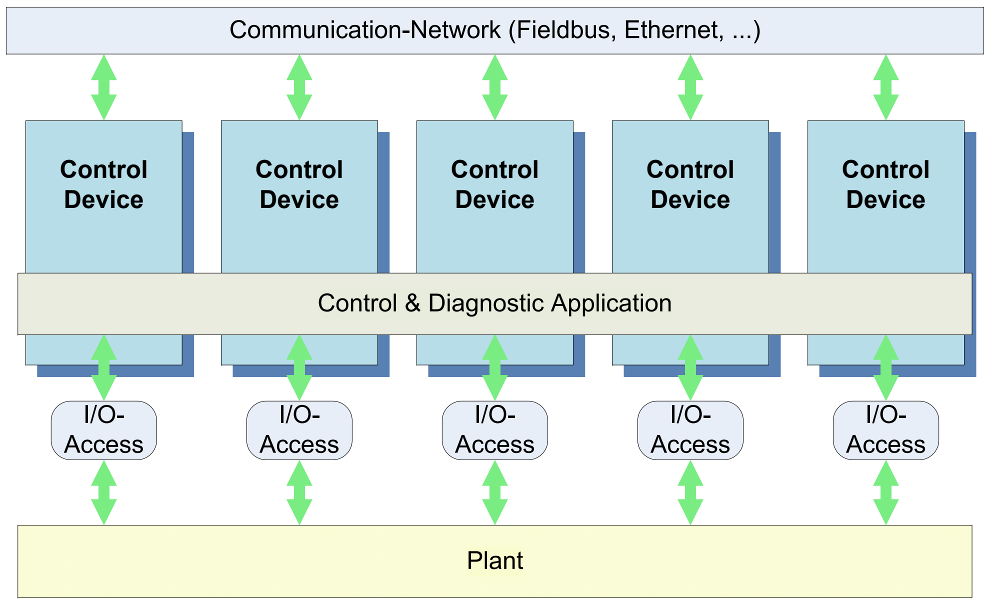


Figure 4.1: Final setup of the automated plant for deployment and testing. The final execution system and the implemented plant have to be available and operational for reliable testing results.

vironment would be endangered due to faults or errors testing may not be applied (e.g. reactor control of nuclear power plant).

During the development cycle of automated plants and automation systems different simulation scenarios may be applied.

Full Simulation Scenario The full simulation scenario is the first to be applied in the work flow. For this approach neither the plant, nor the real execution system has to be available. The control (& diagnostic) application will be generated from the Automation Component Model using previously defined transformation rules (see Section 5.2). These transformation rules have to ensure, that the generated control application (e.g. code) meets the behaviour specification provided by the model. Hence, the interface behaviour of any generated control application shall be replaceable by equivalent implementations. Also the simulation model for the plant is created by model transformation. Without knowledge on the target execution system and desired mapping, the control application is created for a single control device, that is used for HIL-simulation (see Figure 4.2(a)). The control device may be replaced by a fully compatible runtime environment for automatic control (e.g. softPLC). This allows the highest flexibility in simulation time advancement. It is facilitated to run the simulation with increased, decreased, or flexible time advancement (based on a future event list), because the need for time synchronisation is greatly reduced. Later, if information on the execution system

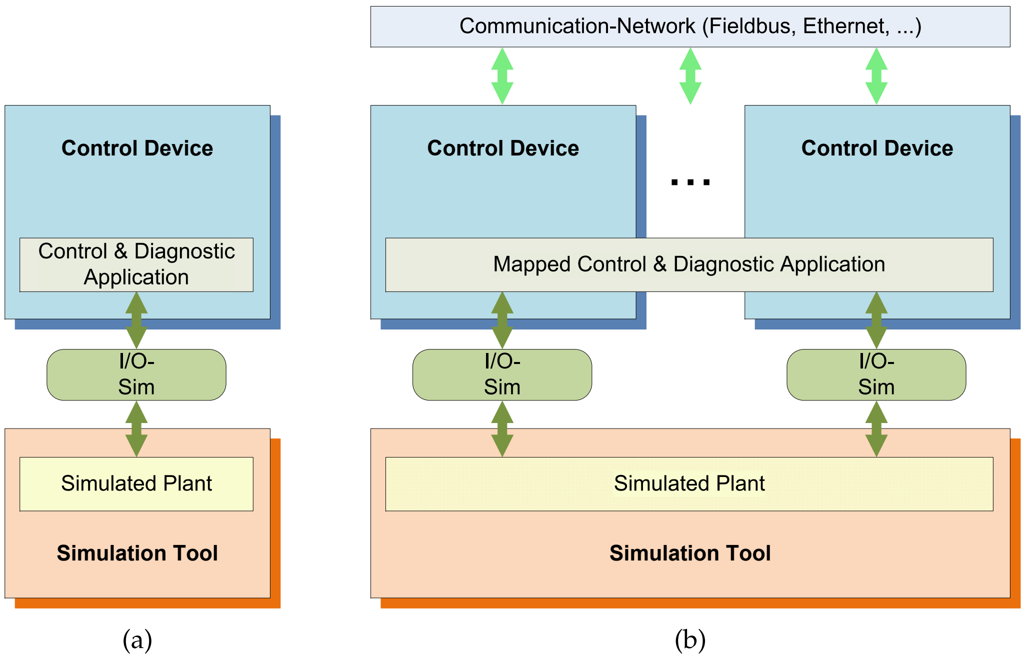


Figure 4.2: Simulation scenarios for the validation of automated control. The plant behaviour is provided by a simulation tool that is coupled to control devices (either physical or virtual) via the I/O-Sim interface. (a) can be used for a coarse validation of the plant behaviour while the execution system specification or mapping information is inexistent or incomplete. (b) shows the simulation scenario with better knowledge on the final execution system setup (including for instance networking).

and the mapping of the applications is available, the number of devices will meet the number in the defined execution system (see Figure 4.2(b)). Also in this sub-scenario it is possible to replace physical control devices with virtual ones (e.g. softPLCs). Hence, also the influence of the mapping decision on the effectivity of the control application can be investigated. Network and fieldbus infrastructure is also covered with this multi-control device scenario. Communication between devices might induce delays, jitter, or overload on the network connections, whose effects are usually only discovered during the commissioning phase.

For some components which are foreseen to be used in the plant, simulation models for specialised simulation tools (e.g. MATLAB/Simulink, Dymola, Modelica) exist. Also these models shall be integrated in the plant simulation, if they are useful for the validation of the plant behaviour. For that reason these simulaton tools need to be integrated in a co-simulation setup. In particular time synchronisation and data exchange between the simulation tools require good knowledge on the involved simulation tools.

The above described scenarios are available early in the design and implementation workflow. No part of the plant nor the control hardware needs to be physically available.

But simulation can be applied during the whole lifecycle of automation systems. Automated manufacturing systems have to be adapted to new requirements by changing their structure. Validation is also needed during reconfiguration phases to ensure the functionality of the target system.

Hybrid Simulation Scenario Hybrid simulation is apt to validate reconfigured or extended plants. The characteristic of hybrid simulation is the combination of simulated components and operational components. The coordination of operational components and simulated components has similar requirements as the co-simulation use-case. Since physically available components operate at nominal time, also simulated components have to use nominal time advancement. Physical interactions of the components (e.g. material flow) have to be considered from case to case. Manual intervention, like replacing manipulated parts, might be necessary [101, 102].

Two extreme scenarios can be identified for hybrid simulation:

- The new component is physically present and operating normally while the rest of the plant is simulated.
- The existing plant is operating normally, while components which shall be added are simulated.

The first scenario can be used for the commissioning and testing of the new components at the developer's site. Figure 4.3 provides an overview of the required simulation setup. The surrounding environment (i.e. the already existing plant where the new component will be integrated) and its interaction with the new components are provided by simulation while the new component is pysically present and operational.

The second scenario is suitable to check the behaviour of the target plant (including extensions) during the specification phase. Different potential solutions for new components can be virtually integrated in the existing plant, as shown in Figure 4.4. The benefit of such a scenario compared to pure simulation is that no extensive simulation models of the existing plant are required. However, the simulation of the new component might be supported by simulated interfaces (e.g. physical, network). Hence, this scenario is mainly supported for the sake of completeness of the proposed simulation framework.

However, in between these two extreme scenarios any combination of operational and simulated components is possible. This is useful to reduce the complexity of interfaces between simulation and operation (e.g. need for manual intervention for part manipulation).

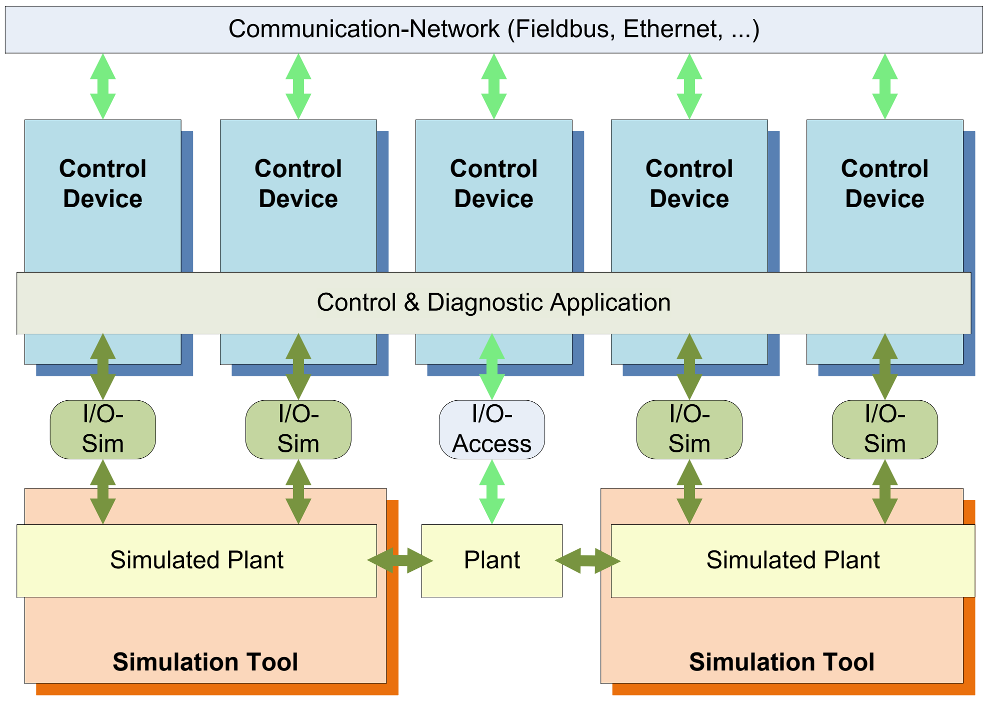


Figure 4.3: Hybrid Simulation for the commissioning and testing of a single components. The control and diagnostic application is executed on the target execution system, consisting of real control devices.

4.2 Plant Behaviour Modelling and Specification

Simulation is a well accepted validation methodology for the specification and implementation phases in multiple disciplines. However, a major requirement for the application of simulation in the automatic control domain is a reliable specification at the interface between the automatic control application and the plant. These interfaces are represented by *I/O-Access* and *I/O-Sim* entities in the Figures 4.1–4.4.

For reliable simulation results the behaviour at these interfaces has to be the same for simulation and operation. The automatic control application must not be able to determine if it is interacting with the simulated plant or the real process. For that reason, the model for the specification of the plant behaviour has to incorporate enough details for such a stateful behaviour description at the plant interface.

4.2.1 Behaviour at the Plant Interface

Knowledge on the expected plant behaviour is essential for the specification and implementation of automated control applications. However, related in-

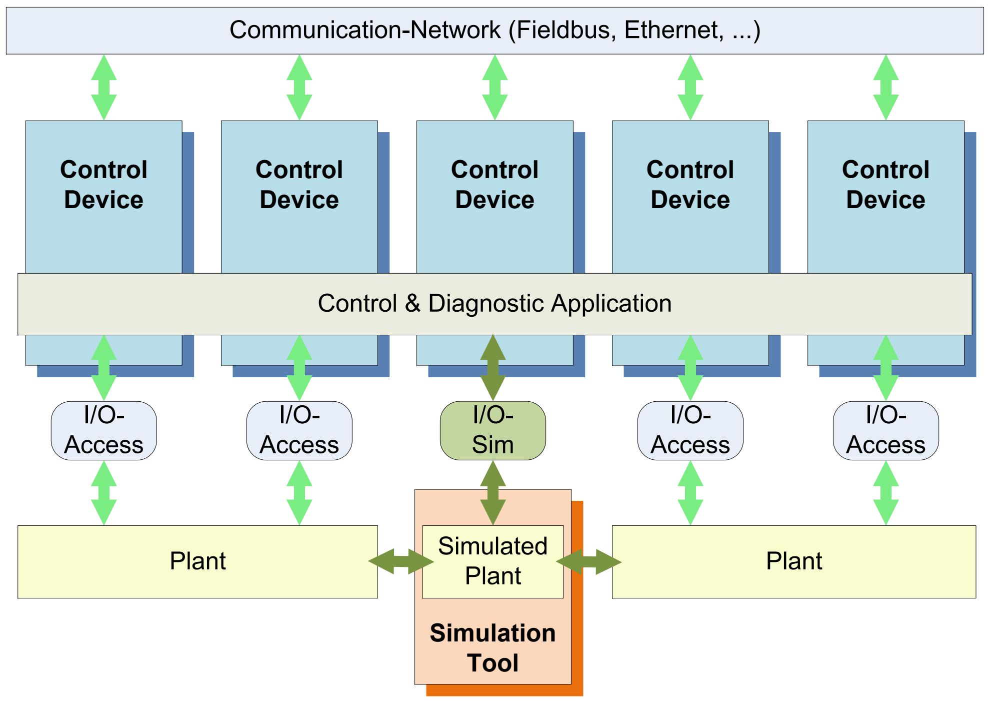


Figure 4.4: Hybrid Simulation for the validation of the integration of additional components into the automated plant.

formation is often provided in non-formal way. To reduce the overall engineering effort the specification shall be made more formalized.

Modern automated control systems are based on digital microcontrollers. This requires that inputs are sampled and discretized. But also the values for the outputs are provided by the automated control system in a time discrete manner. The time intervals for sampling may vary from system to system—from the μs to the s-range (or even larger). The inevitable discretisation at the interface makes a time discrete behaviour description at the interface sufficient. Furthermore, the interfaces between plant and automated control system may have a stateful behaviour. If, for example, the plant is in a fault-state, it will react differently to signals (i.e. outputs) of the automated control system than in normal operation.

The behaviour of the plants at their interfaces can be specified in timed state charts. This specification methodology is able to cope with the stateful behaviour as well as with time discrete inputs and outputs. First the nominal behaviour needs to be captured. In order to deal with deviations in expected timing and sensor values, probabilities and standard variances shall be annotated to the according numbers. The reason for such deviations, which are still within nominal behaviour, can be external or internal to the automated component. Changing voltages due to varying load situations or variable pressure in the pneumatic system are examples for external uncertainty sources. Reduced lubrication or wear out conditions are regarded as possible internal reasons for deviations.

4.2.2 State Chart Diagram for Behaviour Modelling

The plant behaviour model holds information on the behaviour of an Automation Component's mechanical part from the interface point of view.

Harel [132] introduced the state chart diagram as suitable modelling method for reactive systems. Timed transitions and hierarchical aggregation of states are included. However, the presented diagram is a generic concept that has been implemented differently in the last decades. Instead of creating a new, tailored state chart, an already existing model is used as starting point. One of the descendants of the Harel State charts is the UML State Chart Diagram [129]. This diagram type offers a high level of expressiveness and concepts (e.g. on-entry actions, on-exit actions). However, UML also foresees to limit the syntactic expressiveness of the general purpose diagram types with UML profiles [129]. Hence, common UML editors (e.g. the open source editor StarUML [154]) can be used to model the plant behaviour.

Within a single AC concurrently executed states are deemed not necessary for two reasons. First the diagnostic approach is based on the automaton paradigm [155]. Second, the same information can be expressed with additional states. Furthermore, the proposed engineering workflow also foresees model transformation to be used. Model transformation rules can cope with the creation or merging of additional states.

To keep the models smaller and more graspable, hierarchical structuring is obtained outside of the plant behaviour model. The plant behaviour model is embedded in hierarchically structured ACMs. Interaction of the plant components is obtained by *physicalPlantInterface* and *logicalPlantInterface* entities.

4.2.3 Adaptation of Automation Component Models

To reflect the needs for the specification and modelling of the plant behaviour the *Plant* element of the ACM needs to be refined. First the *PlantBehaviour* element of the plant model (presented in Section 3.3.1) needs to be implemented as timed state chart. Second, the modelling of interactions of Plant Components (i.e. physical plant elements of ACs) with other entities in the automated plant (i.e. automatic control, other plant components) has to be enabled.

Plant Component Interfaces

The behaviour of a Plant Component (PlC) is affected by "external" events. An AC's control application is interacting with the PlC based on the specification of the control behaviour. Data and events are exchanged bidirectionally via *PlantInterfaceElement*s. Also physical links influence the overall behaviour of an AC. Since these links are not represented in the control behaviour, they have to be modelled at the PlC level. The types of the physical links are dependent on the domain and the plant. In the primary application domains (i.e. discrete manufacturing and industrial robotics) material flow and kinematics are identified as most important physical links. For each type of physical link providing and accepting ports have to be added to the PlCs' interfaces. The ports represent for example flanges or other connectors. In the containing AC these ports have to be connected to represent the actual physical link.

Plant Behaviour Meta Model

For the specification of the plant behaviour of an AC a state diagram is chosen. Due to the event based interaction of the PlC with its environment (e.g. other PlCs, control application) a simplified version of the UML State Chart Diagram (see [129]) is used as template.

Only a single state can be active at one time. Transitions from one state to another are modelled with *Transition* elements. For the activation of a transition the according transition condition has to be fulfilled. Either data from other physical components or the controlling automation application can be used as guarding condition. Furthermore, also timed transitions (e.g. the time to extract a pneumatic axis) are implemented in the meta model.

Changes of a PlC's internal states are promoted at its interfaces in *Action* elements. Physical interactions as well as changes in sensor values are provided either via the physical interface or the control interface. *Action*s can only be provided for state entry and state exit. *Action*s, which are executed within a state, circumvent the event-based execution. Changes at the interfaces by such an *Action* are not clearly related to an event (e.g. condition at the interface, timed condition). Furthermore, such actions can be replaced by additional states with *Action*s either on state entry or on state exit without loss of expressiveness. Variable changes that would occur within a state (e.g. movement of axis) can be modelled at the state exit, where the environment is notified of the change (e.g. position sensor).

Figure 4.5 is providing the class diagram (meta model) of the State Chart Diagram for the specification of the behaviour at the plant interface. The hierarchical aggregation of PlCs is reached by aggregation of the enclosing AC.

Additional simulation models

Industrial users already apply simulation for selected problems and plants. For efficiency reasons (i.e. avoiding the repetition of the modelling task) such existing simulation models are integrated in the proposed models. Hence, the

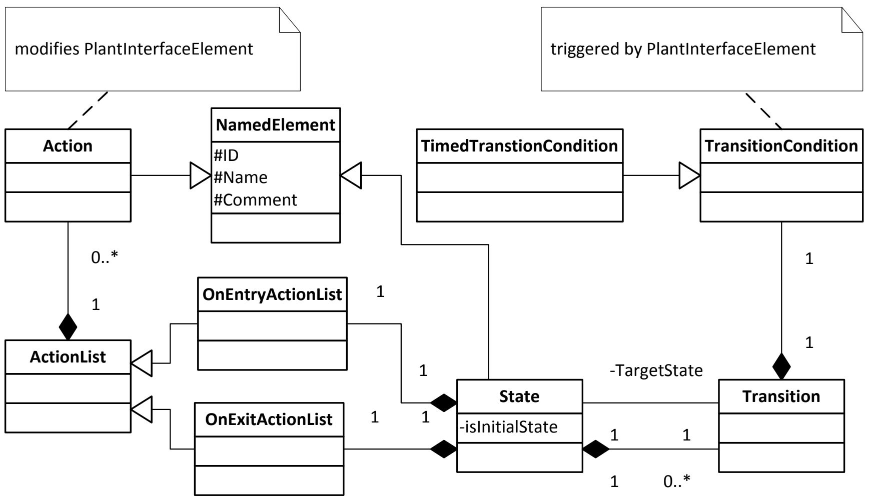


Figure 4.5: Meta model of State Chart Diagram for Plant Behaviour modelling.

plant behaviour of certain ACs is additionally represented by external models. External simulation tools may use different methodologies for the representation of the behaviour. For that reason these external behaviour models are included as additional models in the *PlantModel*. However, these included models are not semantically checked and are only provided to the external tool. Another option is the information extraction from such external models and representation in the provided behaviour model. However, the second option requires additional research, especially a thorough analysis on the models and the underlying concepts for each external tool.

4.3 Aptness of the Generic IEC 61499 Concepts for Discrete-Event Simulation

Industrial automation systems, even if controlling continuous processes, have a discrete character. Hence, discrete-event simulation is a suitable methodology for the validation of automation systems. The analysis of available discrete-event simulation environments (e.g. AnyLogic [74], Enterprise Dynamics [73]) has shown four important properties:

- Events: Discrete events are the central paradigm. They allow variable time advancement and also facilitate the coupling of multiple systems (see Section 2.4).
- Component based modelling: Functionalities are encapsulated modules that are parametrized.

- Hierarchical aggregation: Multiple components can be combined in a new module to model more sophisticated functionalities. Hence, the simulation model remains graspable.
- Clearly aranged simulation models: All interactions of two simulation components can be integrated in a single connection.

However, commercially available discrete-event simulation environments are not capable to directly run and validate automatic control applications. Automatic control engineers will be the main user group for the application of validation to automated control systems. Prevalent knowledge of this user group reinforces the investigation, whether industrial automation runtime environments are apt to be used as basis for a simulation framework.

IEC 61499 sets requirements for industrial control that make it also interesting as basis for simulation execution. The following section investigates the applicability of IEC 61499 based runtime environments for that purpose. General concepts and commonly accepted methodologies are analysed first.

The standard IEC 61499 and compliant runtime environments are inherently apt for distributed execution. The coordinated, distributed execution of automated control applications was a major driving force for its creation. The absence of global variables, the encapsulation of functionalities in function blocks, hierarchical aggregation, and the event based execution mark the most distinguishing changes from previous standards and implementations.

4.3.1 Application-Centric Engineering

A driving force for the creation of IEC 61499 was the increasing complexity of control systems. Multiple control devices, possibly from different vendors, are typically used for automated control in industrial processes. With device centric engineering each of the control devices has to be programmed individually. Coordination of the controllers and communication between the controllers has to be specified and implemented separately on all involved devices. To reduce this effort, a paradigm shift towards application centric engineering has been introduced with IEC 61499. Application engineers can focus on the target functionalities of the control system, instead of the targeted execution system structure. The engineered functionalities are then split and mapped to a system of heterogeneous control devices. Interoperability—a key principle of IEC 61499—ensures that devices from multiple vendors can be used within a single automation application. The second key principle—configurability—provides the possibility to use a single engineering tool to develop and distribute the control applications to these heterogeneous systems. The third key principle—portability—allows the application exchange among various engineering tools.

4.3.2 Events

Industrial automatic control systems react to changes of states of the plant. Therefore most industrial systems (e.g. manufacturing systems, logistic systems) can be represented as discrete event systems [21]. IEC 61499 promotes an event-based execution of the automatic control applications [59]. The reasons for such a shift in execution paradigms—compared to continuous execution of relay systems and scan-based, cyclic execution of previous PLC systems—are diverse. First, the introduction of events allows the specification of the execution order of function blocks within an application [57]. The explicit specification helps engineers during the implementation of automatic control applications for a single automation device as well as for distributed control systems with multiple automation devices [156]. If needed, also scan based execution can be realized with event-based execution, as this execution paradigm is more general [59, 157]. Second, in distributed automatic control environments effects caused by aliasing and jitter are avoided. These can occur if the cycles or clocks) on multiple networked devices are asynchronous and an automatic control application is executed across these devices.

4.3.3 Components

In the IEC 61499 design methodology functionalities are encapsulated in so called Function Blocks (FBs). These FBs are components with well-defined interfaces [158]. In the prevalent modelling languages of the IEC 61131-3 standard, global variables are a central element. For example the I/O access is provided via global variables. Global variables, and thus potential hidden interfaces, hinder reuse of functionalities. Also the multiple instantiation of such components within a single application implies drawbacks and pitfalls. Hence, global variables are prohibited in IEC 61499. Hidden interfaces and invisible interdependencies of functionalities are restricted. FBs well resemble the components of the model-based engineering approach presented in Chapter 3.

Three main types of FBs are defined:

- Basic Function Block (BFB): The behaviour of BFBs is determined by the so called Execution Control Chart (ECC), which is a state machine and algorithms that are executed on state entry. These algorithms can only access internal variables and data which are provided at the FB interface.

- Service-Interface Function Block (SIFB): This type encapsulates functionalities which are provided by the underlying system (e.g. timer, I/O, network). This type can only be treated as a component, if the interface behaviour is well defined and documented [158].

- Composite Function Block (CFB): CFBs are encapsulating other FBs. Thus the functionality is determined by the functionality of the contained FBs and their connections. As long as all contained FBs can be treated as components, also the CFBs fulfil the according properties.

Only SIFBs are able to break the component concept. Also CFBs that contain such SIFBs are rendered non-components. Therefore a closer look is taken at SIFBs.

Service Interface Function Blocks

Access to external data and events is provided by SIFBs. These external data and events include inputs and outputs (i.e. the process interface), hardware or operating system functionalities (e.g. network or timer), and functionalities provided by the runtime environment. This close relationship to services of the hosting control device or process is limiting the instantiation of SIFBs to certain control devices. If services of a different device are needed, the proxy design pattern, which has been adopted by Christensen [159] for IEC 61499, is applicable. Also functionalities of non-compliant devices can be encapsulated in and represented by SIFBs in IEC 61499 based automatic control systems [159]. By such means, external simulation tools can be integrated.

There exist two stereotypes of SIFB: passive requesters and active responders. Requesters are actively triggered from the automatic control application. Responders are triggered by the runtime environment and indicate external events. SIFBs provide and request all data and events via their well-defined FB interface. Automatic control applications explicitly use external functionalities via SIFBs and thus avoid hidden interfaces.

The explicit access via SIFBs is a big advantage of IEC 61499 for hybrid simulation. In hybrid simulation parts of a control application are accessing real I/Os while other parts are interacting with simulated I/Os. Hidden access, for example via global variables, imposes a higher effort in setting up the simulation environment. Such access can be found in IEC 61131 based runtime environments. An intermediate mapping application (decoupling the automatic control application from the I/O access) can be used in such setups to facilitate switching from simulated to real I/Os and vice versa.

4.3.4 Hierarchical Aggregation

In engineering disciplines, like mechanical engineering, hierarchical aggregation (e.g. assembling) of components and assembly groups is known and used for a long time. Standardised and reusable parts and components increase maintainability and reduce costs (e.g. fewer types of parts have to be kept in the magazines, higher volumes of standardized parts can be bought). Hierarchical aggregation has been introduced for the structuring of ACs in order to gain the same advantages (see Section 3.2).

For increased comprehensiveness of automated control applications (i.e. software) the standard IEC 61499 provides means to structure control applications in a hierarchical way. Functionality provided by FBs can be aggregated by two different structural elements: CFBs and Sub-Applications (Sub-Apps). Both, CFBs and Sub-Apps are reusable entities, which can be subject to testing and/or verification methodologies to increase software quality [160, 161]. Although both mechanisms can be used to encapsulate FBs and corresponding event and data connections, they have different properties.

A CFBs provides a well-defined interface, which is static as for any other FB type. Instances of CFBs are units that cannot be split and deployed to different resources. Due to this limiting property CFBs shall only be used to encapsulate well defined functionality and not for structuring purposes [162, 163].

The purpose of Sub-Apps is to group parts of control applications [60]. To other entities in a control application, a Sub-App provides an FB-like interface. However, these interfaces can be modified during application design (i.e. not as strict as FB interfaces). A Sub-App interface can include any interface element of an encapsulated FB or Sub-App. Even more, Sub-Apps may be split and the contained function blocks can be mapped to multiple resources or devices. These properties recommend Sub-Apps for hierarchically structuring applications (i.e. automatic control and simulation) in the same way as in the referring models.

4.3.5 Coupling Entities with Adapter Interfaces

The adapter concept is an elegant way to simplify the definition of FB and Sub-App interfaces [164]. A specific set of event and data inputs and outputs builds up an adapter interface [60]. This common interface is used in FBs and Sub-Apps either as provider (i.e. plug) or as acceptor (i.e. socket) [59]. Plugs and sockets of the same type can then be interconnected by a single "cable"—the adapter connection—which encapsulates all event and data connections provided by the adapter type. The great reduction of connections in automatic control application design, which avoids scattering, is often seen as the main advantage of adapters [59, 164].

However, the strict separation of both sides (i.e. the provider and the acceptor) is also a great benefit. Both components can be specified, implemented, and tested independently from each other. Furthermore, as long as the same interface (i.e. adapter) is used, components can be easily replaced or used in different applications.

Lewis [59] shows the application of the adapter concept for I/O access of automatic control applications. The provided example separates the analogue data processing (e.g. linearization, filtering) from the analogue sensor interaction (e.g. access to the transducer interface). This prevents from direct use of raw values, which might differ from setup to setup (e.g. different voltage ranges).

4.3.6 Summary

The concepts of IEC 61499 (and the compliant runtime environments) provide a suitable basis for the implementation of a distributed discrete-event simulation framework for industrial automation systems, as discrete-event simulation tools use similar modelling and execution paradigms. For co-simulation and hybrid simulation environments the aptness for real-time execution is important. The external events (e.g. timer, network, process) can be directly mapped to events in discrete-event simulation.

The SIFB concept allows a good separation of the process and the automatic control application as well as the implementation of clear interfaces. Such clear interfaces are the key to switch from simulation to operation with little engineering effort. Furthermore, additional data types—extending the data types specified in IEC 61131-3 (e.g. BOOL, INT, LREAL, STRING)—can be easily defined and used in IEC 61499 applications.

4.4 Execution Semantics: Choice of Runtime Environment

Events in event-based industrial automatic control systems originate from external sources. The most important external source is the controlled plant or process. Sensors capture such changes and trigger actions in the automatic control systems. Other external sources include for example the system clock or network interfaces. The controlled process sets the maximum number of events that can occur during a defined period. The automatic control system has to be able to process the process related occurance rate of events. Higher rates (also called event showers) are related to a fault (e.g. broken, flickering sensor) and may lead to an overload of the automatic control system [62].

The implementation of events in IEC 61499 compliant runtime environments is not regulated by the standard. Therefore, multiple variants for the event handling have been implemented by the architects of the available runtime environments. Differences in the execution semantics are relevant for the capability to implement the different world views for discrete-event simulation (see Section 2.4.2). Sünder et al. [136] as well as Čengić and Åkesson [165] have compared execution semantics of different runtime implementations:

- Non-pre-emptive Multi-Threaded Resource: In this execution model, SIFBs act as event sources within the automatic control application. Other FBs which are connected via events are successively called. The execution of the event source (i.e. SIFB) as well as all called FBs is terminated when no further output events are sent by any of the FBs. This implementation reduces the effort for data latching, as the further execution of a once called FBs is blocked until the termination of the event. However, this also causes problems with feedback loops, where at least one FB should be triggered at least twice by the same event source. Also event splitting has been recognised as problematic with this execution model [166]. The best known runtime environment which implements this execution model is the Function Block RunTime (FBRT), which is part of Function Block Development Kit (FBDK) [167].

- Cyclic buffered execution model: All FBs in an application are activated in a pre-defined order. The bigger an application is, the more time is needed to activate all FBs, even if they have not received an input event. This execution model is similar to IEC 61131 scan based execution. Hence, a runtime environment with this execution model is capable to run IEC 61131 applications and event based IEC 61499 applications alike. IsaGRAF has implemented this execution model in its runtime environment [168].

- Event Dispatcher: The third described execution model is using an event dispatcher. Each occurrence of a sent event is put into a FIFO queue. Multiple Event Dispatchers can be used within a resource. This allows to have different priorities (e.g. real-time constraints) mixed in a single resource. Hence, an FB instance might be called by multiple execution contexts (e.g. threads). Therefore, an FB has to fetch its input data when it receives an event, and then executes the according algorithms. The runtime environment FORTE has incorporated the Event Dispatcher method. The execution order of the FBs in an application is solely determined by the events. As input data is latched at the occurence of an input event an FB can also be triggered multiple times as a result of the same external event (i.e. feedback loops are supported). Furthermore, the execution of the control application, started by an external event, can be preempted. This increases the schedulability and allows keeping real-time constraints [169].

Zoitl [169] shows that also with event-based execution real-time constraints can be met. Hence, a carefully designed automatic control application executed on suitable runtime environments (e.g. FORTE) is able to fulfil real-time requirements of discrete-event system simulation in hybrid environments. That and the possibility to implement or adapt execution features are the reasons for the choice of the runtime environment FORTE. Therefore a closer look on the underlying concepts of this IEC 61499 compliant runtime environment is taken.

4.4.1 Event Handling in FORTE

In principle FBs can be considered passive entities in automatic control applications. They remain inactive until an input event is triggering their execution. In the runtime environment FORTE the activation of FBs is done by the event dispatcher.

How are events generated within FORTE?

Some special types of SIFBs—responder FBs—do not stay passive. They are either always active to gather information on the controlled process or are activated by entities different from the event dispatcher. Responder FBs trigger the execution of the control applications. Hence, they are also called Event Source Function Blocks (ESFBs), as all events in automatic control applications originate from them [170]. E_RESTART is cleary recongnizable as ESFB. The missing input events indicate that this FB is generating events independently from the automatic control application, only dependent on the current state of the control device. But also timed FBs (e.g. E_CYCLE, E_DELAY), receiving network FBs (e.g. subscriber, server), or SIFBs providing access to inputs (e.g. sensors, push buttons) are ESFBs. A selection of ESFBs, which are available on all devices, is provided in Figure 4.6.

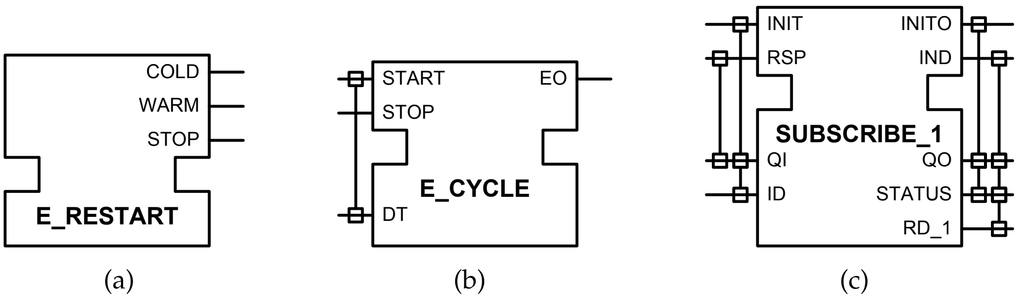


Figure 4.6: Various Event Source Function Blocks trigger events in the application dependent on external events. (a) The E_RESTART FB indicates the start and stop of the control device. (b) The timed FB E_CYCLE repeatedly triggers application events. (c) Network FBs like the SUBSCRIBE_1 FB indicate the reception of data over the network.

Event Chains

Zoitl [169, 170] introduces the event chain concept for IEC 61499 automatic control applications. Automatic control applications have the need for multiple ESFBs. ESFBs are the sources of events within an automatic control application. All further events in the automatic control application are subsequently issued by FBs as a result of the first events issued by the ESFBs. Even more, for any event the original ESFB-triggered event can be identified. All events that stem from the same ESFB-triggered event are part of the same event chain. Multiple event chains may be active at the same time. For real-time execution the different event chains in an application are prioritized and scheduled to meet the previously specified real-time constraints.

Termination of Event Chains

FBNs, and more specifically event chains, represent control algorithms. Determinism of these algorithms is essential, as processes have to be influenced under real-time constraints. Hence, event chains have to terminate, to provide a result, and to allow schedulability. Multiple execution of a single FB within a single event chain (i.e. feedback loop) has to be carefully used to avoid endless loops, which contradict the requirement for termination of the algorithms.

Three ways to terminate event chains have been identified [170]:

- Obviously an FB without output events is unable to trigger further events. Therefore, an event chain is terminated by such an FB.
- Also FBs which do have output events, but where no event connections are attached, terminate the further execution of an event chain.
- The third possibility to end an event chains is a FB, which does not send output events due to an internal state or algorithm (e.g. E_PERMIT). This case imposes the highest effort for the a priori analysis of the event chain termination.

The knowledge on the termination of event chains is necessary for the a priori calculation of execution times and the planning of execution schedules. Hence, real-time execution of applications can be guaranteed on compliant event-based runtime environments (e.g. FORTE) [169].

Handling of External Events

As illustrated earlier, automatic control applications composed of FBs remain passive, until they are triggered by external events. ESFBs are the proxies who bring these external events into the automatic control applications and start event chains. Hence, external events are the reason for all events that occur in automatic control applications. Mainly for reasons of portability the system architecture of the runtime environment FORTE is grounded on a hardware abstraction layer (HAL) [171]. The HAL provides the same interface towards hardware resources (e.g. timer) independently from the concrete hardware platform (e.g. PC running Windows, embedded device running Linux).

As further step towards hardware independence, the resource handlers also provide sophisticated management functionality. FBs register at the appropriate resource handler to indicate their need for a specific service. The manager is able to list those FBs and coordinate the access to the managed service.

The timer handler only requires a single hardware timer, whereas an implementation without such a coordinator would use one timer for each FB which needs timer functionality. This optimization is essential for portable, embedded software, as hardware timers are a scarce resource on embedded systems[5]. Furthermore, the timer handler has a similar functionality as the future event list of discrete event simulators (see Section 2.4.2).

Similar restrictions as for timer access also apply for network connections. FBs waiting to receive data from a communication partner (e.g. server, subscriber) have to either block their execution entity until they receive data or have to regularly poll for received data. Either an execution entity (e.g. task, thread) is needed per receiving FB or execution time is wasted to check for received data. Both mechanisms are expensive in embedded software development. A central network handler is able to efficiently provide the same functionality with only a single execution entity per device [171].

As soon as the ESFBs is initialised, it registers itself, dependent on its own class, at the according manager function. Whenever an external event (e.g. hardware timer notification, received network data) occurs, the manager identifies the appropriate FB and triggers its execution. The notified ESFB translates this external event into an output event and thus starts an event chain. Figure 4.7 provides an overview of the layered architecture of FORTE. Independent of the number of ESFBs or resources within the device, only a single handler per managed functionality is provided. Via the HAL, which virtualizes the hardware and operating system functions in a unified way, the required functionalities are accessed.

The HAL and the resource managers, above all the timer manager, can be adopted to provide time advancement functionalities as they are required by discrete event simulation environments (implementation details are provided in Section 5.3.1). The HAL is also a suitable entity to transparently provide a clock synchronisation service that is required if more than one device are used

[5]Micro-controllers usually provide 2-6 hardware timers, that can be used by application software.

for simulation execution.

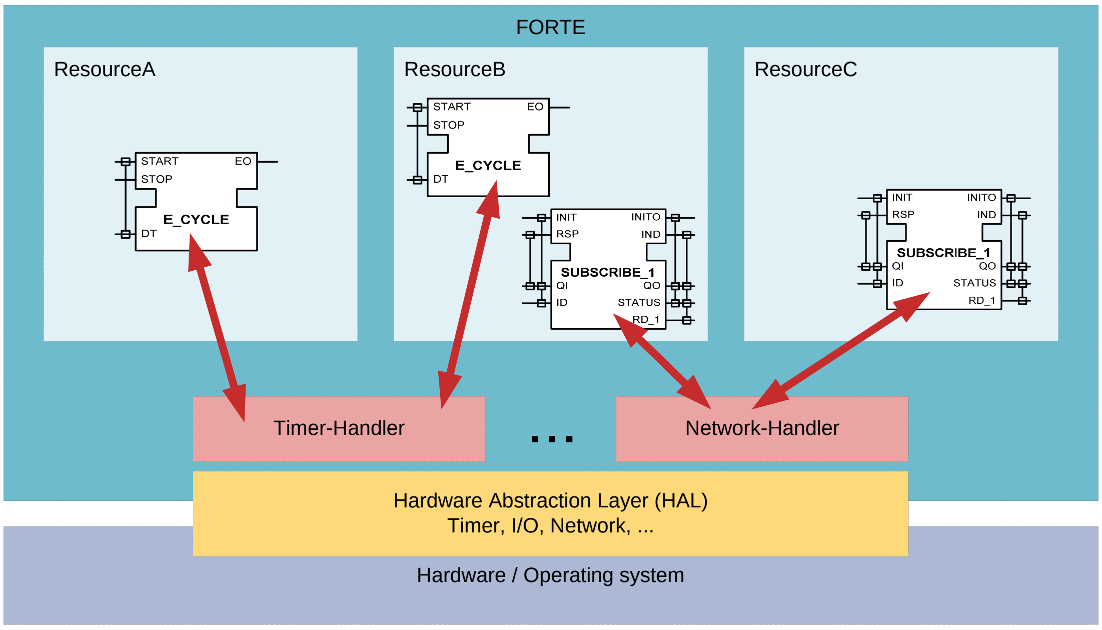


Figure 4.7: Overview of the FORTE architecture. Hardware access is provided via a hardware abstraction layer. Management functionalities, that only exist once per control device, trigger event source FBs on the occurrence of external events.

4.4.2 Summary

FORTE is an IEC 61499 compliant runtime environment. As such it provides all generic concepts of IEC 61499 that are needed for discrete-event simulation (see Section 4.3). Also advanced features of discrete-event simulation frameworks (e.g. variable time advancement) can be realized with the runtime environment FORTE. The central manager for timed FBs allows skipping simulation time to the appearance of the next event (compare to the event scheduling world view in Section 2.4.2). The managed list of timer events is equivalent to the Future Event List of simulation environments. For the scenario involving only a single runtime environment for the execution of the automatic control application and the simulation of the process this is a simple task. For distributed simulation the timer handlers of all involved runtime environments need to be coordinated, which can be transparently handled within the HAL.

4.5 Validation of Behaviour with Event-based Automatic Control Applications

In the previous section we could see the aptness of the automatic control runtime environment FORTE for discrete-event simulation. Hence, FORTE will be the core element of the simulation framework for the validation of the behaviour of Automation Components (ACs). The goal is to deploy the validated automatic control application to the target platform. Therefore, both the control behaviour and the plant behaviour of the AC need to be included in the validation process. However, a clear separation of both aspects is essential. Changes to a validated automatic control application have to be kept minimal, as each change can introduce errors. Therefore, different applications shall be used for the automatic control and the plant simulation. SIFBs in both applications link them and enable the overall simulation.

The plant behaviour simulation has to resemble the (expected) behaviour at the process interface. The control application shall not notice any differences from the operation (with the real plant). Apart from the nominal behaviour—when everything is working fine—also known faults of the plant shall be simulated. Dependent on the automated process, faults may arise with low probability[6]. Using such realistic probabilities could prevent faults to occur during simulation runs. However, simulation is not apt to detect all errors in software specification, design, and implementation [133]. The fault handling for previously unknown plant behaviour cannot be validated due to missing specifications. However, the robustness of the automatic control application and the effectiveness of fault management functionalities can be validated for known or expected faults.

The main application of the presented simulation approach is in model-based automation system development. Both the automatic control application as well as the plant behaviour simulation application has to be generated from the provided model. Model transformation techniques are utilized for that step. Both generated applications shall be deployable without the need for adaptations by control engineers. Nevertheless, these experts have to be able to capture the functionality and identify problems and errors. A clear structure, resembling the hierarchical AC structure, helps to trace errors back to the ACM. For a good visibility of the interfaces, these SIFBs should not be hidden deeply in CFBs.

In the following section guidelines for a structured application design will be presented.

[6] Robot vendors for example state mean time between failures (MTBF) values of up to 80.000 hours (that is more than 9 years in 24/7 operation) for some robotic applications [172, 173]

4.5.1 Structuring Automatic Control Applications

Many different stakeholders get in contact with the automatic control applications during the lifecycle of an automated plant (e.g. component developer, system integrator, control engineer of the plant, operator, maintenance personnel). For this reason—apart from software quality criteria—it is important to provide automatic control applications that are easily graspable by any of these stakeholders. Well-structured applications reduce the effort to review and assess such applications. Confidence in the provided solution—backed by simulation—increases the efficiency of the commissioning phase for new plants or the integration of new components.

In case of breakdowns and errors of the plant, maintenance personnel needs to identify the causes. As automated plants are highly sophisticated, multi-disciplinary applications, also automatic control applications, need to be checked. In this phase scattered control applications can cause longer downtimes, which are potentially costly.

Furthermore, modern automated plants are evolving over the time, as they get adopted to meet new requirements. Also the according automatic control software is changing. Software changes that are necessary to fix discovered errors during ramp-up or maintenance phases need to be represented also in the central data-repository. Hidden interfaces (e.g. global variables, hidden communication channels) have to be avoided. They impede the comprehension of the functionality but also reduce reusability of components using such hidden interfaces.

For all these tasks a clear, comprehensible structure of the automatic control application is useful. This structure has to resemble the hierarchical structure of the ACs. Relationships of parts of the automatic control application with the specification and documentation provided by the ACM have to be clearly visible.

Different concepts—CFBs and Sub-App—are available to implement hierarchical aggregation. Furthermore, hierarchical aggregation is clearly restricting the use of component interfaces. Only connections to a single component in the next higher level and to components in the next lower level are allowed. Especially connections within a hierarchical level are forbidden. This resembles the concept of hierarchical aggregation using sub-components. It also helps to keep interfaces between components well-defined and stable. The component of the next higher level can pass required data and events to other components of the same level (i.e. coordination functionality).

Adapters (and adapter connections) are suitable for these inter-component interfaces, and are proposed for this purpose. Resulting automatic control applications—either generated or created manually—are less scattered. A single adapter connection encapsulates multiple data and event connections (in both directions). Only point-to-point connections between exactly one plug and one socket are allowed and possible. Errors resulting in missing or misconnected data or event connections are prevented by the typed nature of adapters.

From a logical point of view there is no preference, which side of the component interface is represented by plug or socket. Whether the higher or the lower abstraction level shall be represented by plugs or sockets is merely a matter of taste. A flow from left to right, as for writings, can be considered as criterion [163, 174]. Combined with a top-down approach in grasping and understanding things this leads to the following convention: Top-most levels in the hierarchical aggregation are placed on the left-hand side, lower-level implementations to the right. Hence, higher composition levels are represented by plugs and lower composition levels by sockets. The use of Sub-Apps and adapters for the hierarchical aggregation and structuring of automatic control applications is shown in Figure 4.8.

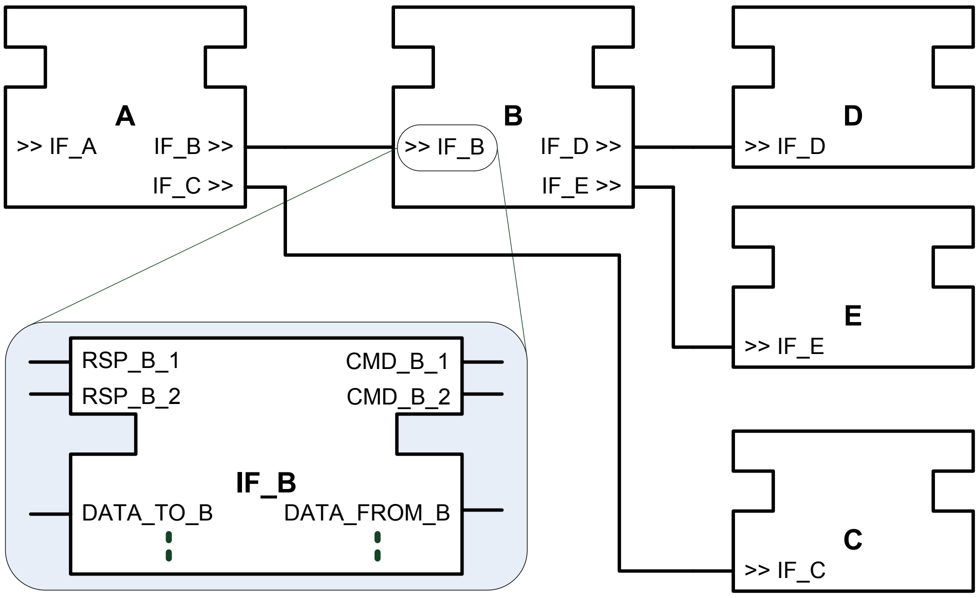


Figure 4.8: Hierarchical Structuring of automatic control applications: The sub-applications (A, B, C, D, E) provide bi-directional interfaces in form of adapters (e.g. IF_B, presented in bottom left). These adapters may include multiple event and data interfaces. By connecting compatible adapters with adapter connections (i.e. connection lines between the sub-applications) all data and event interfaces, that are provided by the adapters, are also connected.

4.5.2 Process Interfaces for Automatic Control Applications

Increasing reusability, and thus reducing development effort, is a driving force in automatic control research. This issue is tackled by the creators of IEC 61499 with the paradigm of device independent modelling of distributed automatic control applications [59]. Finished applications are later distributed (i.e. mapped) to the available control devices. This methodology is also known as application-centric engineering.

However, the mere presence and knowledge of this paradigm is not enough. Exploiting runtime specific mechanisms, which influence the behaviour of automatic control applications, has to be avoided. Also the inclusion of hardware-access functionalities in automatic control applications is rendering automatic control applications hardware specific. Separate steps—hardware-independent development and mapping to specific devices and available I/Os—are often intertwined. A major reason is the limitation in available execution platforms and control devices. Hence, methods and design patterns are needed to increase portability and reusability.

Nested Inputs and Outputs SIFBs which provide access to I/Os can be directly included in automatic control applications [160, 175]. This approach leads to fast results when the targeted execution platform (including for example fieldbuses or internal backplanes) is well known. However, possible usage on different hardware is limited. A review of the automatic control applications regarding the used SIFBs is necessary. Either the same SIFBs can be used or other SIFBs which provide the functionality on the new control platform have to replace the unsupported ones. Engineering tools do not support this re-engineering process well [176]. But what is even worse is, that hardware dependencies can be hidden in CFBs. SIFBs can be used like any other FB. Therefore, they can also be used to define the internal behaviour of CFBs. However, such hidden interfaces inhibit the use of such CFBs on different platforms. All inner FBNs of CFBs included in an automatic control application need to be checked for hardware-dependencies. As a CFB may also include other CFBs this has to be done recursively during a re-engineering process. Hence, only hardware-independent functionality should be included in CFBs [162, 163]. Hardware-dependent functionalities have to be added to the automatic control application and need to be connected to the hardware-independent parts via clearly defined interfaces (i.e. FB interfaces).

However, such a re-engineering process to adopt an automatic control application for a new hardware configuration does not solve the problem permanently. Each platform change triggers a new adaptation process. Even for automatically generated automatic control applications some manual checks need to be done by automatic control engineers.

Generic I/O SIFBs can help to overcome the need to review and replace hardware specific SIFBs if the control platform is changed [176, 177]. SIFBs with standardised interfaces (e.g. analogue input, digital output) have to be provided on multiple platforms. Hence, no major changes to the FBN are needed before they are deployed to another platform. Ebenhofer et al. [176] have shown the feasibility of such an approach on 4 different hardware platforms. Hollow shells are provided for the generic SIFBs—digital input and output as well as analogue input and output. Device-specific code is included during the configuration and porting of the runtime environment FORTE to a specific hardware platform. Hence, hardware dependencies are moved from the automatic control application into the runtime environment. For each runtime environment the effort of porting has to be taken. SIFBs with stable, generic interfaces do not need to be replaced within the application. However, the runtime system needs information, where to write or read process data. Configuration data, which is dependent on the actual implementation (e.g. port, slot, module), needs to be provided as parameters. To hand this information to the runtime system, such data can be encoded into a single parameter input [176]. As a consequence only configuration parameters have to be adopted when moving hardware-dependent SIFBs to another platform. With improved tool support this task is easier to perform than searching and replacing FBs.

Local Multicast Pattern The separation of I/O access from the automatic control application is another approach to increase portability. Thramboulidis [178] introduces the concept of Industrial Process Parameters (IPP). These are a set of parameters of the underlying process, which are used in the automatic control application. They are linked to a Process Interface FB Diagram. The actual I/O access is modeled in this separate application also by means of IEC 61499 FBs. Landsteiner et al. [179] also propose the separation into two applications: *(i)* a control application, and *(ii)* an *I/O* application. Hardware dependent FBs are banned from the control application. SIFBs within the I/O application provide the hardware access. Hardware independent SIFBs are used to link both applications. The Local Multicast design pattern using the PUBL/SUBL SIFBs is proposed in [179]. In the Local Multicasting mechanism data is published (by a PUBL-FB) together with an ID on a common channel within the device (e.g. shared memory). Any potential recipient (i.e. SUBL-FB), who uses the same ID receives the data. This is an 1 to many data exchange mechanism. Replacing these links by network communication (e.g. using UDP multicast) is easily possible. Hence, also different devices can be used for I/O access and control application. Big effort has to be put in the configuration of the communication channels, as the correct IDs have to be used by all communication FBs. Furthermore, a data-type mismatch is only recognized during runtime. Also here, the SIFBs can be hidden within several layers of CFBs.

The industrial runtime implementation of nxtControl, which is based on FORTE, is using so-called *SymLinks* [180]. They are based on the local multicast pattern and automatically provide unique communication channels on the automatic control application side. Hence, only on the I/O access application side the communication channels have to be configured, greatly reducing the effort.

For all approaches based on the (local) multicast pattern the interfaces to the I/Os are hidden within the automatic control application and not directly visible. It is not easily possible to identify the parts of the automatic control application, which are interacting with the process. Hence, grapsing the functionality (e.g. in case of maintenance) is impaired, which potentially prolongs down-times. Even more, these approaches reintroduce global variables, which have been banned for good reasons, in a slightly more regulated form. Potential pitfalls are also multiply connected/used inputs and outputs. Furthermore, data type inconsistencies can be hardly checked.

Instrumentation- and Control-Points The concept of Instrumentation- and Control-Points (ICPs) has been introduced for the abstraction of plant interfaces [181]. ICPs provide—equivalent to measurement points in process control engineering—a unified interface to specific, well-defined parts of the plant (e.g. component, module). All relevant hardware access functionality (i.e. sensors and actuators) are encapsulated by the ICP. Towards the automatic control logic these functionalities shall be represented by adapters. Hence, a clear separation of logical and hardware-access functionalities with a well-defined interface is reached.

For the adapter interface (i.e. the ICP) functional requirements, and to a limited extent also non-functional requirements (e.g. timing), as well as constraints can be defined and visualised. The interaction of the automatic control application and the controlled process can be specified with service sequence diagrams. Timing of the interactions can be specified with extensions to service sequence diagrams. During the hardware independent development of the automatic control application, the hardware access is represented by the ICP (i.e. plug). Within the hierarchical structure of the automatic control application only the lowest levels have access to hardware via the well-defined interfaces of the ICPs (see Figure 4.9).

Another advantage of these well-defined interfaces is, that the ICPs also act as variation points for the concrete hardware implementation. Hardware components that provide compatible implementations together with compliant interfaces towards the automatic control application (i.e. the same adapter type as ICP) are interchangeable. Further requirements for the provided services, that are not be specified for the ICP (e.g. force, momentum, range), have to be taken into consideration though.

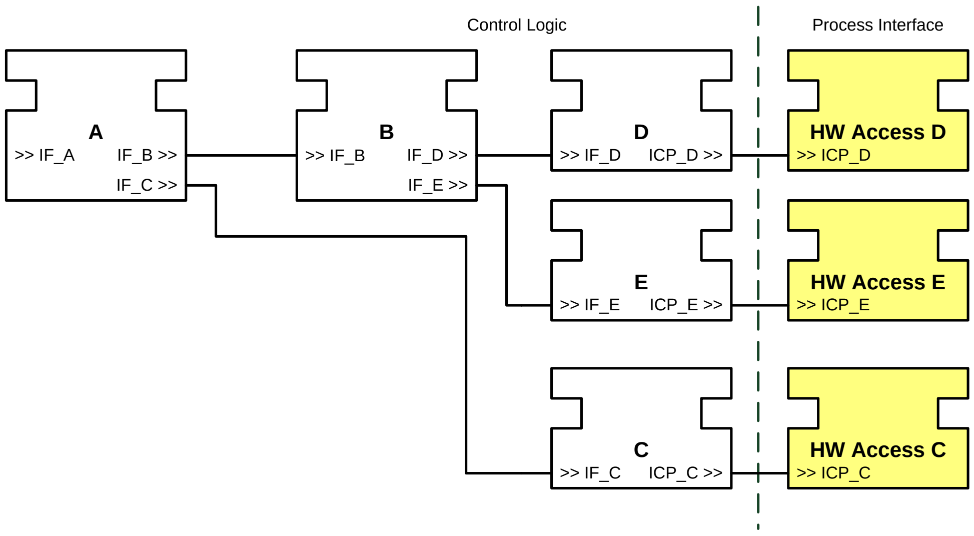


Figure 4.9: The hardware-access functionality is extracted from the lowest control level into special Sub-Applications. These Sub-Applications are connected with adapters, which represent Instrumentation- and Control Points (ICP).

Adaptation of ICPs to Control Hardware and Platform Hardware access functionality has to be provided by the control devices. As interface towards the automatic control application the socket of the ICP is used. Hardware access Sub-Apps encapsulate SIFBs, which either directly access attached equipment or provide access to equipment that is connected via fieldbus systems [164]. The possibility to change the interface (except the adapters themselves) as well as the internal FBN even late in the implementation process (e.g. even after the mapping to the devices took place) is the biggest advantage of Sub-Apps over CFBs.

Additional functionality for the adjustment of sensor values can be added, if the data format of the values received from the hardware interface does not match the expected format at the ICP. If in the automatic control application a temperature value in floating point representation (data type REAL) is expected and the temperature sensor only provides a voltage signal at the hardware interface a value transformation between the two representations has to be added (e.g. based on sensor characteristics or look-up table) [59]. The hardware-access Sub-App can be adjusted during the deployment process to meet the actual situation in the plant (e.g. change of port for a specific sensor). Hardware interfaces that may be used by multiple logical components (e.g. fieldbus access) can be integrated in a single hardware-access Sub-App which provides multiple ICPs.

Finally, also ICPs which provide access to a plant simulation application can be implemented [177]. The explicit connection to either the process interface or the simulation by ICPs on the basis of adapters offers an elegant way to hybrid simulation.

4.5.3 Simulation of Plant Behaviour

The runtime environment FORTE is used as basis for the simulation framework, as it provides concepts that are needed by discrete event simulation environments on the one hand and is able to directly run automatic control applications on the other hand. Hence, also the plant behaviour is simulated with this runtime environment. The simulation application, which mimics the behaviour of the automated plant, has to be provided as FBN. The same structuring principles as for automatic control applications apply also for simulation applications (see Section 4.5.1).

Starting from the hierarchical ACM a hierarchically structured plant simulation application is logical. Hence, in parallel to the automatic control application, which follows the ACM hierarchy, the plant simulation application is created. However, these applications are not independent from each other. In a validation scenario (i.e. with the simulation of the plant behaviour) the automatic control application is interacting with the plant simulation. Sensors and actuators act as coupling elements. In the automatic control application these coupling elements are encapsulated and provided by ICPs. With well-structured applications only the lowest levels in the control layer have direct access to the hardware.

From the plant simulation application these hardware functionalities—sensors and actuators, either directly attached or integrated via fieldbus—have to be provided. But in contrast to the automatic control application sensors have to be treated as outputs in the simulation application and actuators are inputs. Also in the simulation application the interaction with the automatic control application is restricted to the lowest level. Both applications can then be conceptualised as triangles or pyramids. The number of components is decreasing for higher levels (due to the integration of multiple sub-components). The simulation application and the automatic control application are both hierarchically structured and allow interaction with each other only via well-defined interfaces. These interfaces, the ICPs, are located at the lowest level (i.e. the base of the pyramid). Figure 4.10 visualises the integration of both applications for an exhaustive validation of the automatic control behaviour.

Implementation of Timed State Charts The behaviour specification of plant components rests upon timed state charts (see Section 3.3.1). Physical changes of plant components due to services requested by the automatic control layer are affected with time coefficients. In discrete-event based notation delays are sufficient for modelling the process. For the implementation of timed state

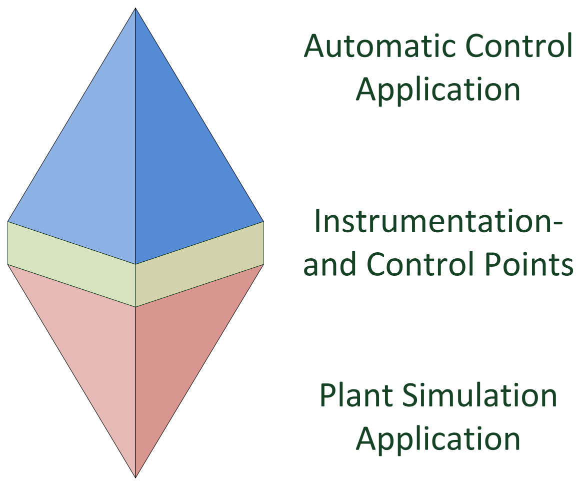

Automatic Control Application
Instrumentation- and Control Points
Plant Simulation Application

Figure 4.10: Automatic Control Application and Plant Simulation are integrated with Instrumentation- and Control Points.

charts with means of IEC 61499 FBs two feasible methods exist:

- New FB type: A new FB type could directly use the timed state chart as specification of the internal behaviour. Hence, the transition from modelling to executing the plant behaviour simulation is simple. However, such an FB type also needs to interact with timers. Timers are provided by the runtime environment (and the underlying operating system or hardware, dependent on the target platform). Such a new FB type is therefore based on the SIFB type. The additional behaviour description method of timed state charts limits engineers in the choice of engineering tools (i.e. violate the configurability principle [60]).
- Existing FB types: The timed state chart (excluding the timing information) can be transformed to an ECC of a BFB. ECCs are finite state machined based on Moore machines. Moore machines do not support exit actions, which are triggered when leaving a state. However, the behaviour specification of the proposed engineering approach (see Section 3.3.1 and Section 4.2.2) makes use of exit actions to keep the specification compact. Hence, additional states need to be added in the ECC. Timed transitions are realised by adding additional input and output events at the BFB. External E_DELAY FBs are connected to these event I/Os and provide timing information to the BFB. Transitions without timing information as well as actions are directly implemented within the BFB. To improve readability, the BFB and the additional E_DELAY FBs are encapsulated in a CFB.

The second implementation alternative is available on different runtime environments and platforms. Furthermore, any available engineering tool, which is IEC 61499 compliant is able to read and modify such implemented FBs. As the state charts of the behaviour specification and the ECC have different semantics, model transformation has to be applied. The first option would allow to directly use the unmodified plant behaviour specification. However, its use would limit the engineers to a single engineering tool. For configurability reasons, the use of existing FB types is the preferred implementation.

Physical Interfaces and Links Physical links are a complimentary concept to the logical behaviour of plant components. Within the physical plant additionally to the data and information exchange also physical exchange takes place. Physical interactions of the components includes for example material transport, forces, and movements. For the simulation of such interactions, physical interfaces have to be added to the components of the plant behaviour simulation application.

The physical exchange is modelled by connections of compatible physical interfaces (i.e. material transport is only possible via the connection of 2 *MaterialFlowPorts*). Even more, the physical interaction is not limited to the same hierarchy as the logical aggregation of ACs.

The physical interfaces of the plant components (e.g. flanges) are represented by adapters. The physical links (i.e. the connection of physical interfaces) are implemented by adapter connections. Adapter interfaces only allow a 1:1 connection, which is equivalent to the real world, physical interfaces. As the plant simulation applications are mostly generated automatically from the specification provided in the ACM, good readability and understanding is ensured. Furthermore, only compatible adapters (i.e. plugs and sockets of the same type) can be connected, which helps to prevent errors.

Currently two types of physical interfaces have been incorporated into the plant model: material flow and kinematic chains for modular robotic applications. These show the feasibility of the proposed approach (see Chapter 5).

Material flow The material flow is an important physical interaction in production systems. Hence, the inclusion in plant simulation approaches is essential. The operation of sensors, which indicate the presence of material (e.g. a palette in a palette transportation system), triggers events in the automatic control application. Material flow between two components is represented by a data exchange in the simulation model. This data exchange has a direction corresponding to the movement of the material from one component to the other. Furthermore, components have a distinct number of interfaces, where they are able to receive material or send it to another component. Each of these interfaces is represented by a *Material Port*—an adapter interface in the IEC 61499 based plant simulation application. Hence, adapter connections are providing the links between two components. These links are logical links only. Such a link between plant components, which represents the material transfer, does not provide space for the material itself. If for example a conveyor belt is completely filled with palettes, it is not able to accept additional material. For that reason, a handshake protocol for the simulated material exchange between plant components is necessary. Hence, material can only be sent to the next component, if this is ready to receive the material.

The internal material handling has to be specified dependent on the type of plant component. A component can represent a buffer for a specific number of pieces (e.g. example aggregating conveyor belts) or change the type of material (e.g. assemble multiple parts to a new part, drill holes in a part). Material handling and manipulation operations have to be specified within the plant behaviour specification.

Kinematic chains For the validation of flexible, reconfigurable manufacturing systems, robotic modules are an important component class. The pose (position and orientation) of a component is dependent on the poses of all previous components in the kinematic chain. Movements of the machines, built from multiple components, require kinematic chains to calculate the absolute position of all components. Interfaces, where components can be connected, are also represented by adapters. The calculation of position and orientation for each connector of a component is calculated using forward kinematics and the geometrical displacements from the component's base (also variable if dependent on actuators provided by the plant component). A widespread methodology for the direct calculation of forward kinematics, based on matrices, has been introduced by Denavit and Hartenberg [182]. The pose at the component's base (in reference to a base coordinate system) is the same as the "output" pose of the (directly connected) previous component. If the pose of a component changes (due to movements requested by the automatic control application), the poses of all downstream components (i.e. towards the end-effector) are recalculated. Since building kinematic chains is a process with an explicit direction, adapter interfaces and adapter connections are well suited as representation in the IEC 61499 based plant simulation applications. Otherwise, multiple data and event interfaces as well as according connections are needed, which results in scattered applications.

Simulation coordination For some components or plants detailed simulation models exist. The reuse of these models for the validation of the automatic control applications could faster lead to simulation results, which are reliable and trusted. Existing know-how on the external, specialised simulation tools (e.g. Arena, Dymola, OpenModelica, Matlab/Simulink) can be used as extension to the behaviour specification via timed state charts, which is the core specification method in the ACM[7]. Nevertheless, not all aspects (e.g. continuous, internal behaviour) can be described with this method. Furthermore, the external simulation tools and their models are well known to domain specialists. However, the integration of external simulation tools requires some adaptation of the simulation models and the coordination of the simulation execution:

- Interfaces between the external simulation application and the plant simulation application (i.e. IEC 61499 based application) have to be included. The purpose of such interfaces is the provision of a control-interface within the simulation application. Hence, access to (virtual) sensors and actuators has to be provided. Three possibilities for the implementation of such interfaces (events and data) exist. First, if the external simulation tool already provides the possibility to exchange data over the network the en- and decoding can be implemented in FORTE. Second, standard network interfaces as provided by all IEC 61499 runtime environments can be used. This requires additional functionalities (e.g. data encoding) to be implemented for the external simulation tool. Third, an independent communication platform (e.g. communication middleware) can be used. Adaptations on both sides are necessary in this case. The choice of the integration method is mainly dependent on the external simulation tool and its interfacing capabilities.

- Synchronous time is a prerequisite, when multiple simulation execution systems are used in the same simulation scenario. Dependent on the possibilities of the external simulation tools and the scenario the time advancement method has to be chosen. For hybrid simulation (i.e. parts of the automated control system are already operational) it is necessary that all simulation tools are running at normal speed (i.e. at world time). In pure simulation mode, with an external simulation tool included, the conservative time synchronisation mechanism can be used, if it is supported by the external simulation tool. This would allow advancing at variable speed. In all other cases, a constant time advancement has to be used throughout the whole simulation scenario. Either nominal time, increased or decreased time advance can be used to meet the simulation requirements.

4.5.4 Execution System

The ICPs previously introduced represent the plant interface in automatic control applications. The hardware access functionality, either hardware de-

[7]The focus lies on the interface behaviour, which can be sufficently modelled with timed state charts.

pendent process interfaces or coupling logic to the plant simulation application in form of SIFBs, is connected to the control logic via typed adapter connections.

Integration of automatic control and simulation execution in a single device Using the same runtime environment as basis for the execution of the automatic control application and the plant simulation application makes it even possible to run both applications on the same devices (see Section 5.4). The adopted simulation scenario is shown in Figure 4.11. The restriction of *I/O-Sim*

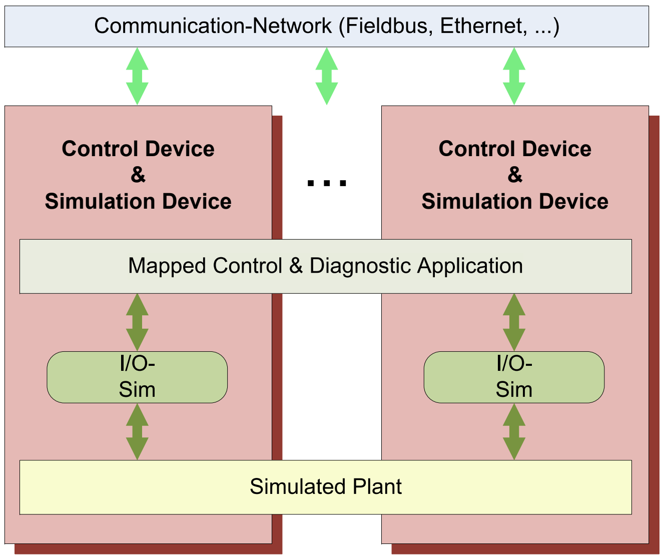

Communication-Network (Fieldbus, Ethernet, ...)
Control Device & Simulation Device
...
Control Device & Simulation Device
Mapped Control & Diagnostic Application
I/O-Sim
I/O-Sim
Simulated Plant

Figure 4.11: The automatic control application and the plant simulation application are executed on the same devices.

interfaces to single devices opens the possibility to limit the hardware access functionality for coupling the automatic control application and the plant simulation application to the local multicast pattern. The local multicast pattern allows to link applications or application parts in different resources on the same device. It is comparable to variables which are globally accessible within a single device. Hence, the naming of the variables (i.e. the channel ID) has to be well chosen to avoid multiple usage of the same link.

Interaction of Plant Behaviour Simulation and Automatic Control Application Generic hardware access FBs can be used in the "normal" hardware access Sub-Apps. Such generic hardware access FBs can then be replaced with local multicast mechanisms by the runtime environment. The configuration parameters of the generic hardware access FBs can then also act as unique ID for the local communication channel. However, the runtime environment has to know, when the real hardware access or the simulation access needs to be used. A manual replacement of the hardware access Sub-Apps is possible. This allows the application of the simulation approach also with runtime environments and engineering tools which cannot be adapted to perform the changes automatically. However, an automatic process helps to reduce the engineering effort for switching from simulation to operation. Four possibilities for automatically changing the target of the generic hardware access FBs are feasible:

- Special device type: With the introduction of a special device for the execution of plant simulation special versions of the generic hardware access FBs for linking automatic control with simulation can be included in the device. The modification of the HAL completely takes away the need to modify the automatic control application. The switching between simulation and operation is only dependent on mapping to the according target device. Through the modification of the HAL it is not possible to access the real process from such a special device. Hence, it is not possible to perform hybrid simulation with only one device.

- Special resource type: It is also possible to provide two implementations for the hardware access FBs in a single device. The decision, which version—direct hardware access or simulation access—can be done according to the resource type. Modifications to the device manager functionality of the runtime environment are necessary. With such a modified runtime environment it is possible to perform hybrid simulation on a single device. Only the change of mapping is necessary to switch between normal operation and simulation. Furthermore, the same control device can be used for normal operation of the plant, without further adoptions to the runtime environment.

- Additional Boolean flag for the generic hardware access FBs: The switching process can also be implemented in the generic hardware access SIFBs. Either as an additional data input (e.g. a flag) or integrated in the parameter input the mode selection can be communicated to the generic FB. During its initialisation phase the selected behaviour is selected. This approach allows mixing access to the real process and the simulated process within a single resource. Changing the parameters is only slightly less effort to replacing the hardware access Sub-Apps. The generic hardware access FBs are more complex, since code for both cases (i.e. access to hardware and access to simulation) needs to be provided. However, in contrast to replacing FBs in a running application, the change of parameters and re-initialization of FBs is easier (i.e. requires less reconfiguration commands) [62].

- Deployment mechanism of engineering tool changed: Instead of modifying the runtime environment the replacement may also be managed by the engineering tool. During the deployment process a different FB type may be instantiated in the runtime environment. This can be done transparently for the automatic control engineer.

Time Management For simulation environments the time management is a core feature. It allows adopting the time advancement to the activity in the simulation model. Hence, the simulation time can advance independently from the world time either at decreased, increased, or at variable speed.

Conservative time management maintains a common list of future events (comparable to the list maintained by the time manager of FORTE). Time is then advanced to the next event in this list. Optimistic algorithms require the implementation of sophisticated roll-back mechanisms, as the causality may be violated (see Section 2.4). Such a feature is specific to discrete event simulation. The integration in an existing runtime environment for automatic control requires extensive changes in the software architecture. Hence, the conservative time management methodology is best suited for the integration in the existing runtime environment.

In a single instance of the runtime environment FORTE the functionality of the future event list is provided by the timer handler. Even networked FBs—another class of ESFBs—only provide (passive) communication services, if they are used within the boundaries of the validated system. Therefore, all events are originating from timed FBs.

However, for the determination of activity within the device—multiple event chains may be active in parallel—and for the coordination of multiple runtime environments the time management functionality is implemented in the event chain and the timer manager classes.

Evaluation of Network Influence In the design of discrete and distributed control applications network communication between the devices has an important influence. Nevertheless, this influence is hardly considered in the design of automatic control systems [183]. Communication systems between the controllers are often treated as ideal at the time of automatic control application engineering. Packet loss, jitter, limited bandwidth, like many other properties of real communication systems are not taken into consideration. For specific fieldbus types configuration tools provide analysis features. Such tools are mostly aiming at scan-based systems with periodic data exchange, for example time-triggered communication protocols [157]. However, worst case assumption can also be used for event-based communication. Nevertheless, tools for the configuration and analysis of fieldbus and network systems are not integrated in the automatic control application design. Feedback is not provided during the development of the application and the mapping decision. Hence, the influence can be experienced for the first time during the commissioning phase. Changes to the mapping, the introduction of more powerful communication systems, or other measures to reach a reliable automatic control system at such a late point in time are costly.

Network simulation has emerged and is currently quite mature. There exist commercial network simulation tools and reliable, academically used and developed open source simulation environments alike (e.g. OPNET Suite [184], ns-2 [185], ns-3 [186]) The mentioned simulators already provide a huge variety of models. Hence, they are applied for the simulation of global backbones (internet) as well as for local wireless applications [186].

Integrating network simulation in the simulation scenarios for the validation of automatic control applications provides feedback on design decisions in a more timely manner. Changes in the mapping and the execution system can be thoroughly planned and alternatives can be evaluated. For that reason network simulation is included in the simulation framework for distributed control systems.

4.6 Summary

Simulation is a validation method that is seldom used during the development of automatic control applications for manufacturing systems. Commercially available simulation frameworks do not allow to thoroughly validate the later deployed automatic control applications. Three scenarios that need to be covered by validation methods for automatic control have been identified: full simulation, hybrid simulation, and testing of the commissioned plant.

The plant behaviour specification is also included in the ACM, which is the central engineering data repository (see Chapter 3). Timed state machines are fulfilling the requirements for the behaviour specification. Also physical interactions of the plant components are included, which directly influence the behaviour of automated plants. From the specified plant behaviour a plant simulation application is generated with similar means as the automatic control application in a hierarchical structure.

As available simulation frameworks are not suitable to validate automatic control applications the aptness of event-based automatic control runtime environments compliant with the international standard IEC 61499 is evaluated. The execution semantics of the runtime environment FORTE makes it suitable as basis for a simulation framework. Event based execution is a more general approach than the scan-based execution [59, 157]. Therefore, it is possible to emulate scan-based execution on event-based runtime environments. For that reason, it is feasible to validate the behaviour of control applications, which have been engineered with the model-based approach presented in the

Chapter 3. Wenger [187] provides strong evidence, that functional equivalence of event-based and scan-based automatic control applications can be reached. Hence, appropriate and well tested model transformations are sufficient for a reliable validation of automatic control applications using only event based execution.

Using the same runtime environment for the operation and the validation is a conveniant solution. This allows to use the same automatic control application (without the need to change) in both simulation and operation. Furthermore, real control devices can be used also in simulation scenarios as soon as they are available. Hence, the final execution system setup, including the communication network, is already used and assessed before the commissioning phase. Both, dedicated simulation devices as well as special simulation resources in normal devices can be provided for any platform that is supported by FORTE. The time management functionality is a necessity for the full-simulation scenario. Implementation options for its integration in simulation devices, based on FORTE, are presented.

For both applications, plant behaviour simulation application as well as automatic control application, a clear presentation of the functionality to discipline experts (i.e. automatic control engineers) is necessary. Guidelines for the creation of hierarchically structured applications using strict interfaces (i.e. adapters) are presented. ICPs are introduced to represent the plant interfaces. These also facilitate switching from accessing the simulated plant to the real plant.

Finally, modelling elements for the implementation of plant simulation applications are presented. FBs, whose behaviour is represented by timed-state charts, are used as representation of plant components. Also methods to represent physical interactions of plant components (e.g. material flow, movements) are provided. These modelling elements, as well as the application guideline, build the basis for the implementation of (semi-)automatic model transformation facilities for the application of code generation.

CHAPTER 5

Implementation and Experiments

In order to validate the modelling and simulation concept presented in the previous chapters all relevant elements of the engineering and simulation framework have to be implemented. Hence, the modelling infrastructure from discipline and domain specific engineering tools to the common Automation Component Model is ellaborated. A second aspect is the automatic code generation from the Automation Component Model by means of model transformation facilities. This process is demonstrated for the target language IEC 61499. Similar model transformation rules are used for both the automatic control application and the plant simulation application.

The proposed development approach offers the possibility to validate the designed plant setup early in the development process. Special focus is put on the integration of simulation as validation methodology.

Finally, test cases from three different application domains are presented: discrete manufacturing plants, robotic applications, and process technology. These test cases show the feasibility of the proposed model-based engineering approach and the integration of various simulation scenarios in the model-based engineering environment, from full simulation via hybrid simulation to full operation as presented in Section 4.1.

5.1 Developed Modelling Infrastructure

In this section implementation details regarding the modelling of plants are presented. This includes

- the integration of discipline specific engineering tools,
- the integrative Automation Component Model, and

- the model transformations to gather information from the engineering tools and bring it into the Automation Component Model.

5.1.1 Integration of Modelling Tools

There is a large number of tools which are used by domain experts to specify plants. With the proposed modelling approach relevant engineering data is extracted from such tools and combined in the ACM. Based on the selected test cases two different tools are integrated for demonstration:

- StarUML: StarUML is an open source UML-Editor [154]. It supports different diagram types, which can be limited or extended by UML profiles.
- Schunk ViroCon: ViroCon is a configuration tool for robotic applications built of components from the company Schunk [188]. It supports the provision of text-file based BoMs as well as movement specifications.

StarUML

StarUML is an open source UML Editor [154]. The software is able to provide many useful functionalities which are also provided by commercial UML tools. It allows to define and use UML profiles. Hence, it is a well suited candidate to show the integration of a Unified Modeling Language (UML) tool in the modelling workflow.

The structure of component based plants can be described with Class Diagrams. The Class Diagram is one of 7 structural UML diagrams, usually used for software development. It is best suited to describe the entities (classes), their relationships, and their interfaces [189, 190]. Only little adaptations are necessary for an automatic interpretation of the model, including hardware information. The behaviour specification of the automated systems can be provided by Statechart Diagrams (see also Section 4.2.2). From the available diagram types two UML diagram types have been chosen:

- UML Class Diagram, and
- UML Statechart Diagram.

Existing UML concepts are used to represent and model building blocks of automated systems. There is no unique way for linking UML concepts to modelling elements, as UML is a general purpose modelling language [189]. Therefore, a semantic fundament has been created with industrial experts in form of a UML profile [189, 191]. It allows unambiguous interpretation of the models and also automatic data extraction. The enriched UML diagrams are a powerful and expressive Domain Specific Language (DSL) for the modelling process of Automation components.

Structural design of the automated systems The UML Class Diagram, which has been concretized by the UML profile, is well suited as DSL for the specification and description of the building blocks of component-based automation systems. A component type (i.e. Automation Component (AC)) is represented by a *Class* in the diagram. Two stereotypes that ease modelling and data extraction have been introduced:

- Class stereotype *Basic*, and
- Class stereotype *Composite*.

Basic ACs have a direct process interface, which enables them to interact with the plant. However, they are not allowed to contain further ACs. *Composite* ACs on the other hand consist of other ACs. The contained subcomponents are represented as classes, which are linked to the composite AC by a *composition*. Cardinality measures indicate the number of contained elements of the same type. Composite ACs are not allowed to directly interact with the plant and therefore use services that are provided by the contained components.

For modelling interactions within a composite component, all contained components are modelled as distinct *Objects* in the Class Diagram. Interactions are represented by *Links* between these objects.

Interface design for a component type The Class Diagram also offers concepts that can be used for the specification of interfaces [189]. UML *Operations* are used for that purpose [191]. Stereotypes have been introduced to distinguish the different interface elements of an AC:

- *IN*: is an inter-component interface element, where the component is receiving information from another component.
- *OUT*: This stereotype marks an interface element, which provides information for other components.
- *PlantPort*: is an interface element that provides the process interface. It is also used to link the plant simulation model with the control model.
- *MaterialFlowPortIN* and *MaterialFlowPortOUT*: These stereotypes designate physical ports of the controlled plant, where material can be received from or provided to other components.
- *KinematicPortIN* and *KinematicPortOUT*: UML operations with these stereotypes represent physical connectors of the plant, where components can be joined.

Modelling control behaviour The behaviour of composite ACs is defined by the included components. For basic ACs the behaviour needs to be explicitly defined. Apart from the model of the component structure (including the interfaces) UML is also apt to specify the behaviour of components. The UML Statechart Diagram is well suited to describe discrete behaviour.

For the modelling of control behaviour the expressiveness of UML Statechart Diagrams is limited by the defined UML profile [191]. Only Moore machines shall be supported by the modelling language. Hence, only entry actions of the states are considered for the specification of the component behaviour. For a deterministic behaviour, one of the states is designated as initial state by the UML pseudostate "initialstate" together with a transition to this state. States and transitions do not require any further extensions or limitations by the UML profile.

Within entry actions, which are executed whenever a state becomes active, interface elements can be changed. The *Tagged Value* concept of UML profiles is used to define the syntax of such changes. *PortChange* is the according concept, which has been defined for the specification of the component behaviour.

State transitions are in most cases dependent on a condition, which has to be met. Such a condition may either be a value of an interface element or a time delay. Each case is designated with a stereotype on the *Transition* element: *PortCondition*, or *DelayCondition*.

Modelling plant behaviour Statechart diagrams are also used to specify the plant behaviour. Plant behaviour and control behaviour modelling is done in separate diagrams. A stereotype *PlantModel* is used to distinguish plant behavour specifications from other diagrams. The same semantics as for the control behaviour specification are applied. However, different interface elements are used to interact with other components.

The PlantPort elements are used to interact with the related control application. Physical ports, such as kinematic ports and material-flow ports, are connected to compliant ports to model physical interaction of the ACs. Changes to the data, for example coordinate change as a consequence of component movement or material handling, have to be modelled in the entry actions of the states.

ViroCon

The ViroCon tool allows to specify and validate robotic structures composed of robotic components from Schunk [188]. A comprehensive catalogue of available components, including information on mechanical connectors, boundaries of movement, and 3D models, is supporting the specification. The workspace of the configured kinematic structure can be calculated and visualised.

Furthermore, movements of the components can be simulated, collisions are detected, and motion plans can be created.

The list of components is provided as XML file. The created motion plan, with movements of the single components, is provided as text-file with a fixed structure.

5.1.2 Developed Models

The Eclipse framework offers an open and extensible infrastructure [192]. Hence, it is used for many software development projects. The Eclipse Modelling Framework (EMF) project uses the Eclipse Framework as basis and provides extensions [193]. EMF provides a powerful meta-modelling infrastructure. Furthermore, comprehensive editing functionalities are provided, which allow the manipulation of the models as well as a tight integration in model-based engineering approaches [193].

These properties make EMF a suitable basis for the implementation of the models presented previously (see Sections 3.3 and 4.2). All meta-models of the proposed models have been implemented as graphical ecore diagrams. The resulting ecore class diagram for the Plant Model (PlaM) is shown in Figure 5.1.

Additional classes have been added to enable model management functionalities. Many elements have been implemented as named elements. Furthermore, the separate models, which represent different aspects of the specified automation project (e.g. control behaviour, plant behaviour), need to be integrated. The top-element in the integrating Automation Component Implementation Model (ACIM) is the *AutomationProject* element. Within this unified model, links between different modelling elements can be established (e.g. *PlantPort* elements). These links allow traversing through the whole model.

The implemented models have been included in a stand-alone application, the MEDEIA Engineering and Configuration Environment (MECE) [194]. MECE provides basic functionality for specifying, reviewing, and refining automation component models. The information is presented in a tree-based editor, which resembles the class relations of the modelling elements. For the PlaM both the *PlantComponentInterface* and the *PlantBehaviourModel* have to be provided. Figure 5.2 provides an insight on the model editor for an example of a *PlantComponent*.

The basic modelling environment is provided as open source project at SourceForge [194, 195]. Additional functionalities, which are needed for the proposed engineering workflow, can be directly integrated in this tool. Data-import and code-generation facilities can be provided as Eclipse plug-ins with full access to the models.

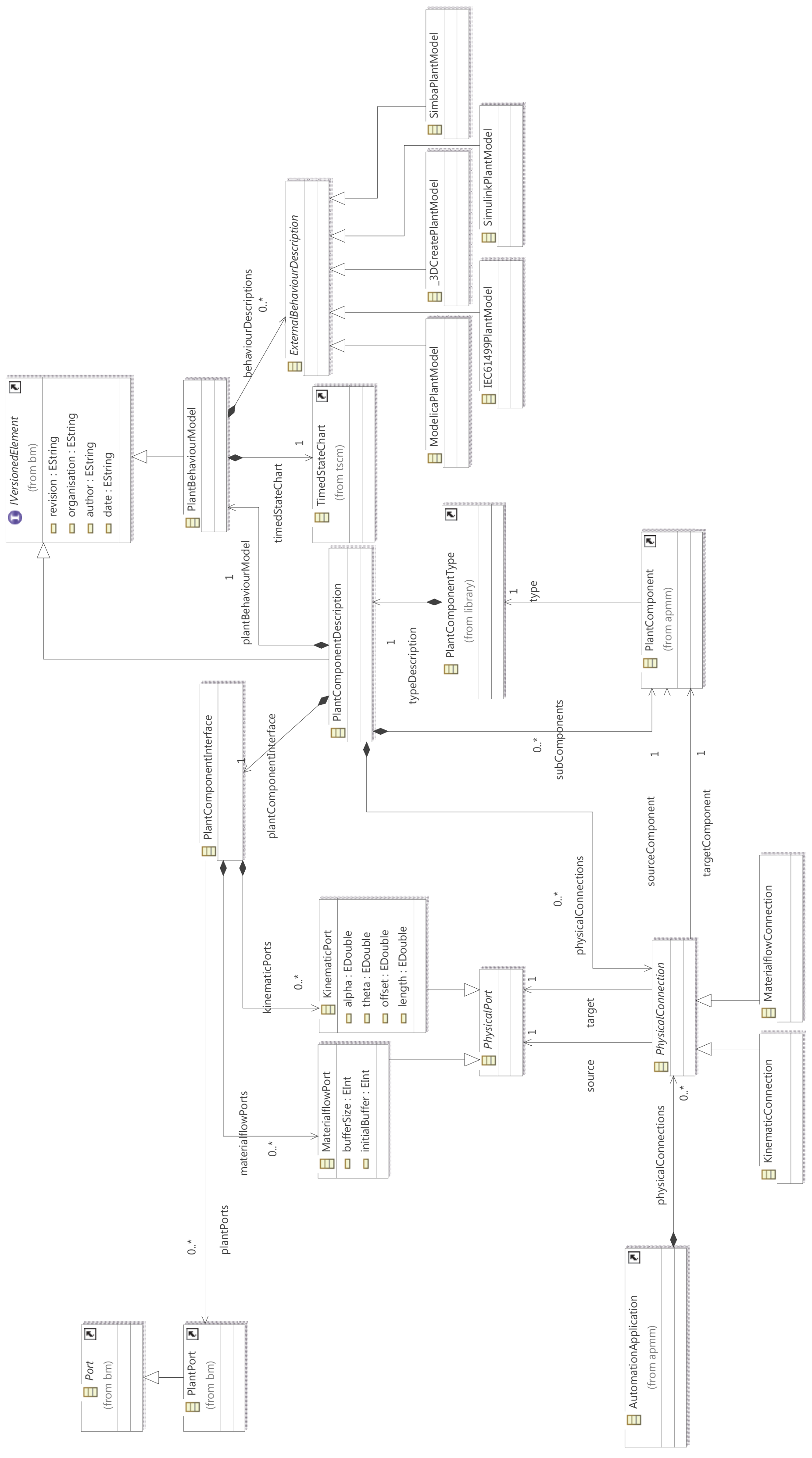
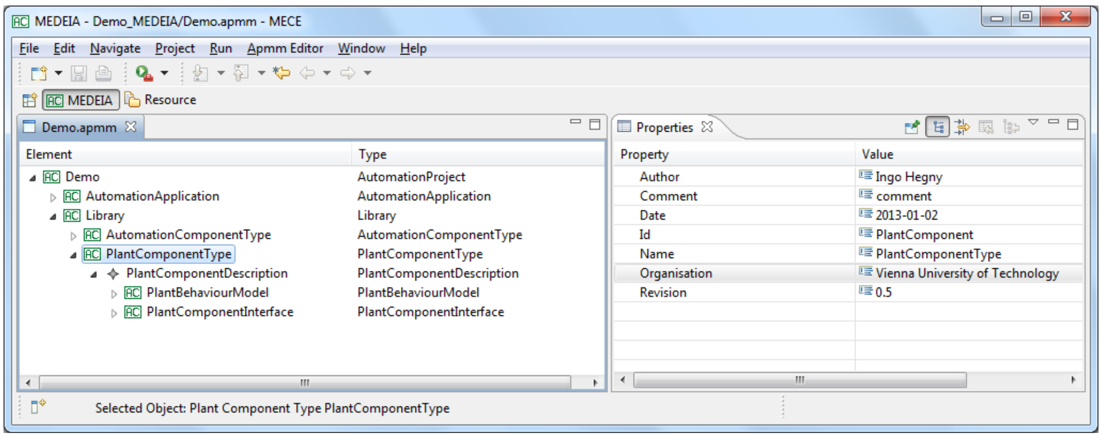


Figure 5.1: Screenshot taken from EMF showing the implemented ecore diagram as meta-model for the Plant Model.


MEDEIA - Demo_MEDEIA/Demo.apmm - MECE
File Edit Navigate Project Run Apmm Editor Window Help
MEDEIA
Resource
Demo.apmm
Properties

Element	Type
Demo	AutomationProject
AutomationApplication	AutomationApplication
Library	Library
AutomationComponentType	AutomationComponentType
PlantComponentType	PlantComponentType
PlantComponentDescription	PlantComponentDescription
PlantBehaviourModel	PlantBehaviourModel
PlantComponentInterface	PlantComponentInterface

Property	Value
Author	Ingo Hegny
Comment	comment
Date	2013-01-02
Id	PlantComponent
Name	PlantComponentType
Organisation	Vienna University of Technology
Revision	0.5

Selected Object: Plant Component Type PlantComponentType

Figure 5.2: Model editor for specifying, reviewing, and refining the automation component model.

5.1.3 Data Extraction

An important step in the proposed engineering workflow is the extraction of relevant engineering data from the discipline and domain specific engineering tools. StarUML with the proposed UML profile follow a well-defined meta-model. Hence, it is possible to implement a model-to-model transformation to gather data from the well-defined StarUML model and integrate it into the modelling environment of the central engineering tool. A two stage approach for the data extraction has been implemented:

1. The XML-based file format of StarUML is parsed and saved into an intermediate model, which can be randomly accessed.

2. A model-to-model transformation facility is used for filtering and transfering relevant data into the respective model in the ACIM.

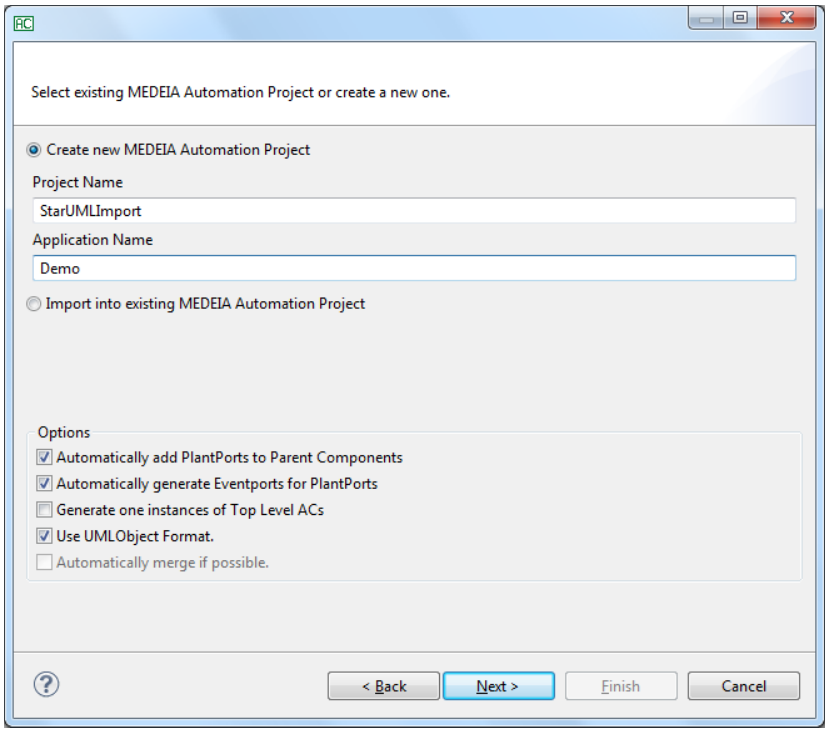
The syntax of XML allows to hierarchically structure data. Such data can be presented as tree, the Document Object Model (DOM). Within this model each element is of the same class. Strings (e.g. tags, values) are used to distinguish the elements. The text-to-model transformation adds a classification of the model elements. A hierarchical model of UML entities—represented as Java-objects—is the result of this step.

In the second step, the hierarchical model is interpreted syntactically and semantically. The previously defined stereotypes provide the necessary semantic frame. Information gathered from selected UML models (i.e. Class diagram and Statechart diagram) is relevant for the Behaviour Model (BehM) as well as the PlaM (see Table 5.1).

The transformation process is implemented as PlugIn for the abovementioned MECE tool. The import wizard, whose user interface is shown in Fi-

Table 5.1: UML diagram to ACIM entity mapping.

UML diagram	**ACIM sub-model**	**ACIM model element**
Class diagram	BehM and PlaM	Only the content of Class diagrams is imported.
Statechart diagram	BehM	Behaviour Specification
«PlantModel» Statechart diagram	PlaM	Plant Behaviour Specification


Select existing MEDEIA Automation Project or create a new one.

(●) Create new MEDEIA Automation Project

Project Name: StarUMLImport

Application Name: Demo

( ) Import into existing MEDEIA Automation Project

Options

- [✓] Automatically add PlantPorts to Parent Components
- [✓] Automatically generate Eventports for PlantPorts
- [ ] Generate one instances of Top Level ACs
- [✓] Use UMLObject Format.
- [ ] Automatically merge if possible.

Figure 5.3: User interface of StarUML import wizard.

gure 5.3, is provided as Java class. Hence, the import wizard can directly access the models via model manipulation functions provided by the MECE tool.

Equivalent modelling elements, which are contained in the Class diagram, are created in the according models (see Table 5.2). After creating the automation component types, the behaviour specification is imported. This can either be a Statechart diagram for basic components or instantiated and connected sub-components for composite components.

The statecharts defined in separate UML Statechart diagrams are used to specify the behaviour of either the control behaviour or the plant component. A common meta-model *TimedStateChartModel* is used. Hence, the same import facilities and mappings can be used. These are shown in Table 5.3.

Table 5.2: UML Class Diagram entity to ACIM entity mapping.

UML Class	**ACIM sub-model**	**ACIM model element**
«Composite» UMLClass	BehM	Composite Automation Component
	PlaM	Composite Plant Component
«Basic» UMLClass	BehM	Basic Automation Component
	PlaM	Basic Plant Component
«IN» UMLOperation	BehM	Inter Component Interface of Automation Component
«OUT» UMLOperation	BehM	Inter Component Interface of Automation Component
«PlantPort» UMLOperation	BehM	Process Interface of Basic Automation Component, attribute direction used for distinction of sensors and actuators
	PlaM	Process Interface of Basic Plant Component, attribute *direction* used for distinction of sensors and actuators
«KinematicPortIN» UMLOperation	PlaM	Pose at the base of the Plant Component
«KinematicPortOUT» UMLOperation	PlaM	Pose at interface to other Plant Components (multiple ports per Plant Components are allowed)
«MaterialFlowPortIN» UMLOperation	PlaM	Receiving Materialflow interface element
«MaterialFlowPortOUT» UMLOperation	PlaM	Outgoing Materialflow interface element
Links	PlaM or BehM	connect linked interface elements of sub-components within containing composite component

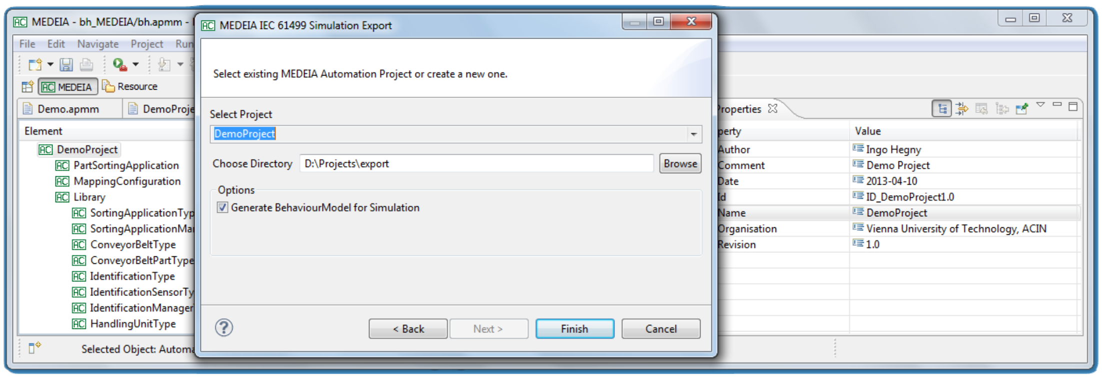
Table 5.3: UML Statechart entity to TimedStateChart entity mapping.

UML element	**ACIM model element**
State	State
InitialState	State with attribute *InitialState*
OnEntryAction	Algorithm
«PortCondition» Transition	Transition with link to an interface element of the containing component
«DelayCondition» Transition	Transition with timed condition

The presented mappings of UML entities facilitate the fully automatic information extraction from UML diagrams. Automation Components specified with StarUML can be brought into the developed ecore models without loss of information.

5.2 Generation of Control and Simulation Code

Like the data extraction functionalities, code generation functionalities can be integrated in the Eclipse-based modelling tool. The wizard for the generation of the the simulation application is depicted in Figure 5.4. The generation of the related automatic control application is selectable via a checkbox in the user interface. The presented wizard is providing Model-to-Model (M2M) and Model-to-Code (M2C) transformation. The provided model transformation framework as well as the workflows for the generation of FBs, automatic control applications, and simulation applications are presented in this Section.



Select existing MEDEIA Automation Project or create a new one.

Select Project: DemoProject

Choose Directory: D:\Projects\export

Options

[✓] Generate BehaviourModel for Simulation

perty	Value
Author	Ingo Hegny
Comment	Demo Project
Date	2013-04-10
Id	ID_DemoProject1.0
Name	DemoProject
Organisation	Vienna University of Technology, ACIN
Revision	1.0

Figure 5.4: Export-Wizard for the creation of the simulation application.

5.2.1 Model Transformation Framework

The source models for the model-transformation are the previously defined ecore diagrams (i.e. the ACIM and its sub-models) that are provided by the extended MECE. For a M2M transformation also a meta-model of the target model is needed. The according meta-model for IEC 61499, which is provided in Backus-Naur Form (BNF) [60], has been translated to an ecore model in the MEDEIA project. This allows to automatically generate Java-classes and manipulation methods for all IEC 61499 modelling elements (e.g. devices, resource, FBs, connections).

Although the transformation algorithms can be implemented directly as Java program, a more efficient way has been chosen. The Modeling Workflow Engine (MWE) is provided specifically for model manipulation and transformation tasks [196]. The generic representation of the workflow, which is used for the creation of automatic control applications from a specified ACIM (for a specific AC), is shown in Figure 5.5.

The process steps within the workflow are defined in XML syntax. Metamodels, which are used within this step are explicitly specified. Each step in

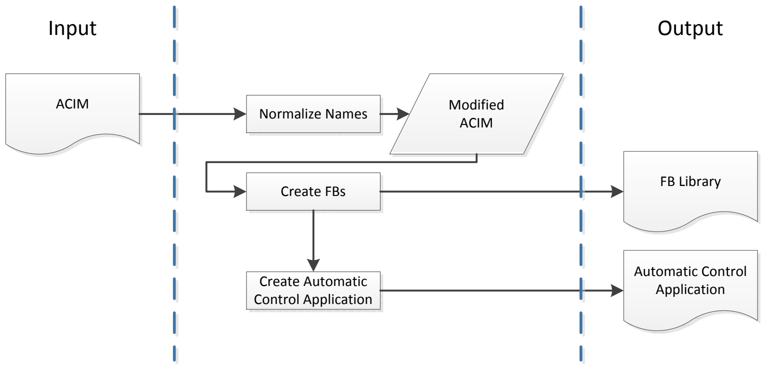

Input
Output
ACIM
Normalize Names
Modified ACIM
Create FBs
FB Library
Create Automatic Control Application
Automatic Control Application

Figure 5.5: Workflow for the generation of automatic control applications (i.e. control code).

the workflow has one output slot, which receives the results of the model-transformation or model-manipulation steps. Finally, model-transformation functionality is called (with the *invoke* command), and the according input models are provided as parameters. The workflow element, which is normalizing the names within the input model, is provided as example in Listing 5.1. It is the first step in the implemented workflow (depicted in Figure 5.5).

Listing 5.1: Process Element for the Normalization of Names in Model Transformation Workflow.

```
<!-- change MEDEIA model for further steps -->
<component class="org.eclipse.xtend.XtendComponent">
 <metaModel id="ecoreMM"
     class="org.eclipse.xtend.typesystem.emf.EmfMetaModel">
  <metaModelPackage value="${medeiaMM}"/>
 </metaModel>
 <invoke
     value="adaptModel::mmBuild::changeModel(medeiaModelOrig)"
     />
  <outputSlot value="medeiaModel" />
</component>
```

The process steps within the workflow are implemented with a domain specific language for the model transformation—XTEND with model transformation extensions. This specialized high level specification language features often needed methods. Iterations over multiple siblings in the model, for example, can be easily performed by MWE and XTEND. In general purpose languages, such as Java, the same tasks would require more complex control structures and recursive methods. Hence, the domain specific language leads to better readible transformation functions, which is an important reason for the introduction of DSLs in general [50]. The iteration to adapt the names in ACs to meet requirements of IEC 61499, for example remove whitespaces or unsupported characters, is shown in Listing 5.2. This is the method, which is called from the workflow element previously presented.

Listing 5.2: Iteration over all Components in an AutomationProject.

```
apmm::AutomationProject changeModel(apmm::AutomationProject
   ap):
  let apn = ap: // Duplicate Automation Project
  apn.applications.automationComponents.changeBMcompos()->
     //Change Control Part of Components in separate method
  apn.applications.plantComponents.changePMcompos()->
     //Change Plant Behaviour Part of Components in separate
     method
  apn; //return changed Automation Project
```

5.2.2 Generation of IEC 61499 Modelling Elements Library

The next step for both the automatic control and the plant simulation is the creation of the FB library. A basic set of modelling elements (e.g. adapters, FBs), which are a requirement for the implementation of specific functionalities modelled in the ACIM, is automatically added to the library. However, neither automatic control nor simulation applications can be automatically built only from pre-defined modelling elements. The behaviour of ACs (plant and control) can be defined by arbitrary statechart diagrams. Furthermore, interfaces of ACs may be freely defined during the specification phase. Hence, new FBs and Adapters have to be included in the library. These are automatically deducted from the interface and behaviour specification of the ACs. For both applications distinct FBs providing different functional aspects are created.

Especially the simulation of physical interactions of the defined ACs requires FBs and Adapters. Currently two kinds of physical interactions are implemented throughout the modelling and simulation workflow: kinematic chains and material flow.

Kinematic Chains

The implementation of kinematic chains is facilitated by Adapters (i.e. *KinematicPort*), which are added to the interfaces of the Plant Components. But also the calculation of Denavit-Hartenberg parameters [182] at the physical connectors is a functionality which is needed multiple times. Hence, an FB *KinematicCalc* is provided for the calculation of kinematic transformations (i.e.

Matrix calculations). It accepts the pose at the base of the component as well as the displacement (e.g. extension, rotation) to the kinematic connector (e.g. flange), where the next component or end-effector is located (see Figure 5.6(b)). Kinematic chains (i.e. forward kinematics) are calculated from the base point towards the end-effector in a uni-directional manner.

If multiple branches are connected to the same base, kinematic parameters have to be shared. Adapters, which are used to implement the kinematic chains, only allow one-to-one connections [59, 60]. This limitation stems from the possibility to exchange data in a bi-directional way. Data inputs may only receive data from a single source. A so-called fan-in (i.e. multiple input connections) would lead to ambiguous values at the data inputs. However, the limiting factor given by the bidirectional data exchange, is not needed for the calculation of kinematic chains. Hence, an FB *KinematicSplit* is provided, which allows to connect two branches to the same kinematic base.

The interfaces of the abovementioned Adapter and FBs are presented in Figure 5.6.

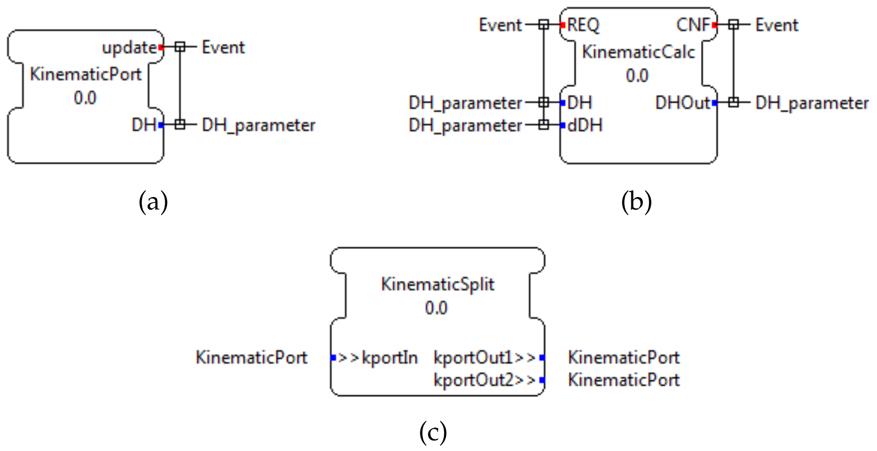


Figure 5.6: Provided Adapter and Function Blocks for modelling Kinematic Chains. (a) Adapter Interface. (b) SIFB for the matrix calculation of kinematic transformations. (c) FB for providing kinematic parameters to two kinematic branches.

Materialflow

The simulation of the materialflow in an automated plant also requires a set of Adapters and FBs.

FIFO-queues for material with configureable size (FB *MaterialflowQueue*, presented in Figure 5.7(c)) are provided for the implementation of input and output queues of ACs. For the manipulation of a single piece of material within an AC the *MaterialHandling* FB is added to the library (see Figure 5.7(b)).

More complex scenarios (e.g. putting together two or more pieces to a single new piece) can be implemented with the according behaviour implementation of the plant component. Multiple pieces are received at the according MaterialflowInput ports and only a single piece is put into the output queue.

The *MaterialflowPort* Adapter acts as interface element within an AC for accepting material from and providing to other ACs. It implements a bi-directional data flow. In one direction the data on the material is provided. In the other direction an indication if material is accepted by the receiving component (e.g. full queue) is provided. The interface of the implemented IEC 61499 Adapter type is shown in Figure 5.7(a).

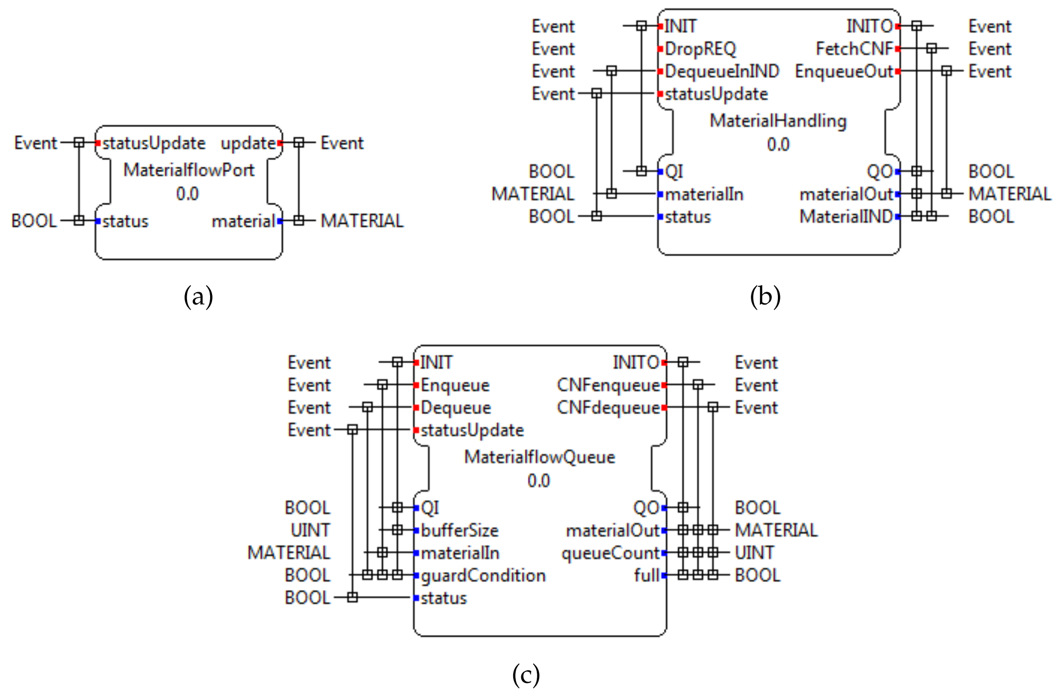


Figure 5.7: Provided Adapter and Function Blocks for modelling Materialflow. (a) Adapter Interface. (c) FIFO Queue with variable size. (b) FB for the manipulation of material within ACs.

Other Required Function Blocks

The generated library of IEC 61499 modelling elements shall be sufficient to model and deploy the automatic control and simulation applications. Apart from the IEC 61499 modelling elements that are required for the implementation of physical interactions of ACs also other common FBs (e.g. E_DELAY) are added to the library.

5.2.3 Generation of Modelling Elements

For the automatic control and simulation of ACs specific FBs are generated and later added to the library. During the generation process the hierarchical design guideline is applied, which was elaborated in Section 4.5 and which is shown in Figure 4.9. Inter-component interfaces are implemented as Adapters. However, Sub-Apps are currently, to the author's best knowledge, only supported by a single engineering tool, the 4DIAC-IDE [197]. Hence, CFBs are used to implement the ACs. Due to the high granularity and separation of concerns with the use of structured interfaces (i.e. Adapters) this change is not restricting. Sub-Applications can still be used within supported engineering tools to reduce the number of visible modelling elements (comparable to folding mechanisms in programming environments) [60].

The generation process is a two-step process. First the interface elements are generated. This step is the same for basic and composite ACs. Then the FB, interface and internal behaviour, is generated, based on the specification in the ACIM.

Interface Elements: Adapters

Adapters are a good way to structure the interfaces of FBs. Multiple data and event connections can be encapsulated within a single adapter connection. Furthermore, adapters are typed modelling elements of IEC 61499. This means that only sockets and plugs, which are compatible (i.e. the same type), can be connected. Hence, less scattered automatic control applications are the result.

The interfaces of FBs that implement ACs are mainly reduced to adapters. Each FB provides its interface elements to higher hierarchical levels in a single plug. Interfaces to specified sub-components are integrated in form of sockets of the according sub-component type. Hence, a single connection is sufficient to connect two ACs in a hierarchical structure. Even more, as long as alternative implementations of sub-components provide the same interface (i.e. plug) they can be used without modifying upper hierarchical levels (i.e. top-down development is supported). To facilitate a gradual deployment of the automatic control application only interface elements needed for the initialisation and deinitialisation are provided in addition to adapter interface elements.

The creation of adapter types, based on the specified component interfaces, is a requirement for the generation of FBs. Hence, adapters for inter-component interfaces as well as plant interfaces are added as first generated elements to the library.

Behaviour Implementation: Function Blocks

Basic ACs and Composite ACs need to be treated differently. The behaviour specification of Basic ACs is based on a timed state chart model. Furthermore Basic ACs are also the only point of interaction between controlled hardware and automatic control application. Only this type of AC may contain ICPs. The behaviour of Composite ACs is solely defined by the combination and interconnection of sub-components. Physical interactions of plant components are also limited to the basic building blocks. Composite plant components are only a logical structuring method, which only exposes the physical interfaces of contained sub-components.

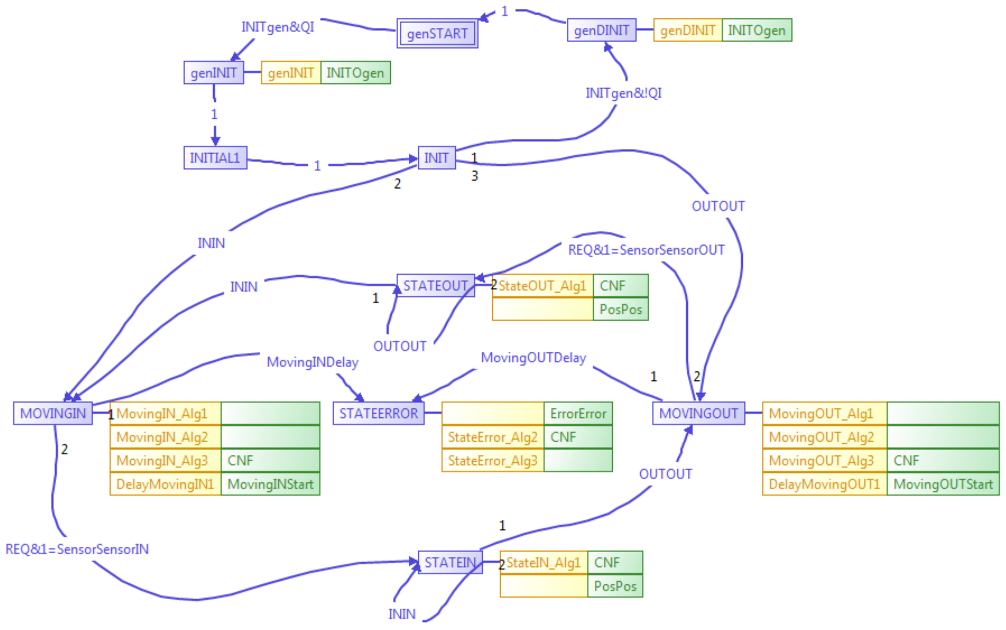
The generation process for the modelling elements for both automatic control and simulation is structured similarly, as both follow exactly the same AC structure. However, since different models and entities are used, a separate set of transformation rules is provided.

Basic Automation Component Basic ACs are characterised by two properties:

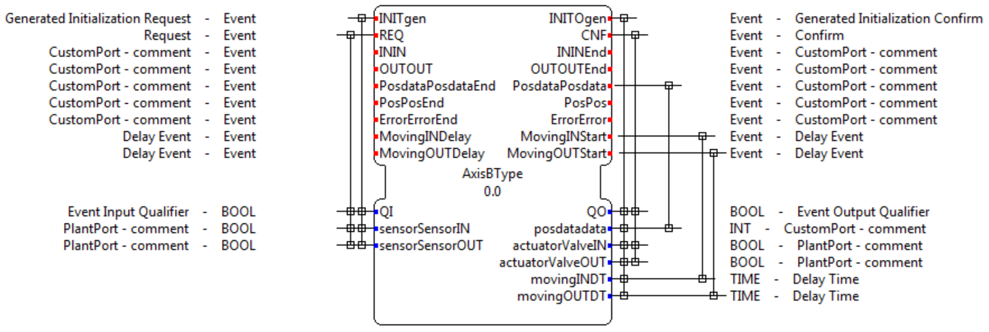
1. their behaviour is specified by timed state chart diagrams, and
2. only they can provide interactions to the process.

The most portable way to implement timed state chart diagrams with available mechanisms of IEC 61499 is to transform the state chart without timing information into an ECC of a BFB (see Figure 5.8(a)). Timed transitions are implemented as event-triggered transitions. An additional event output—to trigger the start of the transition—and also an event-input, which indicates that the specified time has elapsed, are added to the interface of the BFB (see Figure 5.8(b)). Within the encapsulating CFB (see Figure 5.9) per timed transition an E_DELAY-FB is added and connected to the additional event in- and output (see Figure 5.9(a)). The socket, which provides the interface to higher hierarchical levels, as well as the plug, which provides the ICP, where appropriate, are also accessible within the encapsulating CFB. To allow the interaction of the behaviour implementation (i.e. the BFB) with higher hierarchical levels and also the physical process the interface of the BFB provides all interface elements as data and event inputs and outputs.

Hence, two FBs are added to the library for Basic ACs without process interaction. As soon as a Basic AC also includes plant interface elements a third FB, also a CFB, is provided as encapsulation of the hardware access FBs. A suitable ICP is used to model the interface between the CFBs.



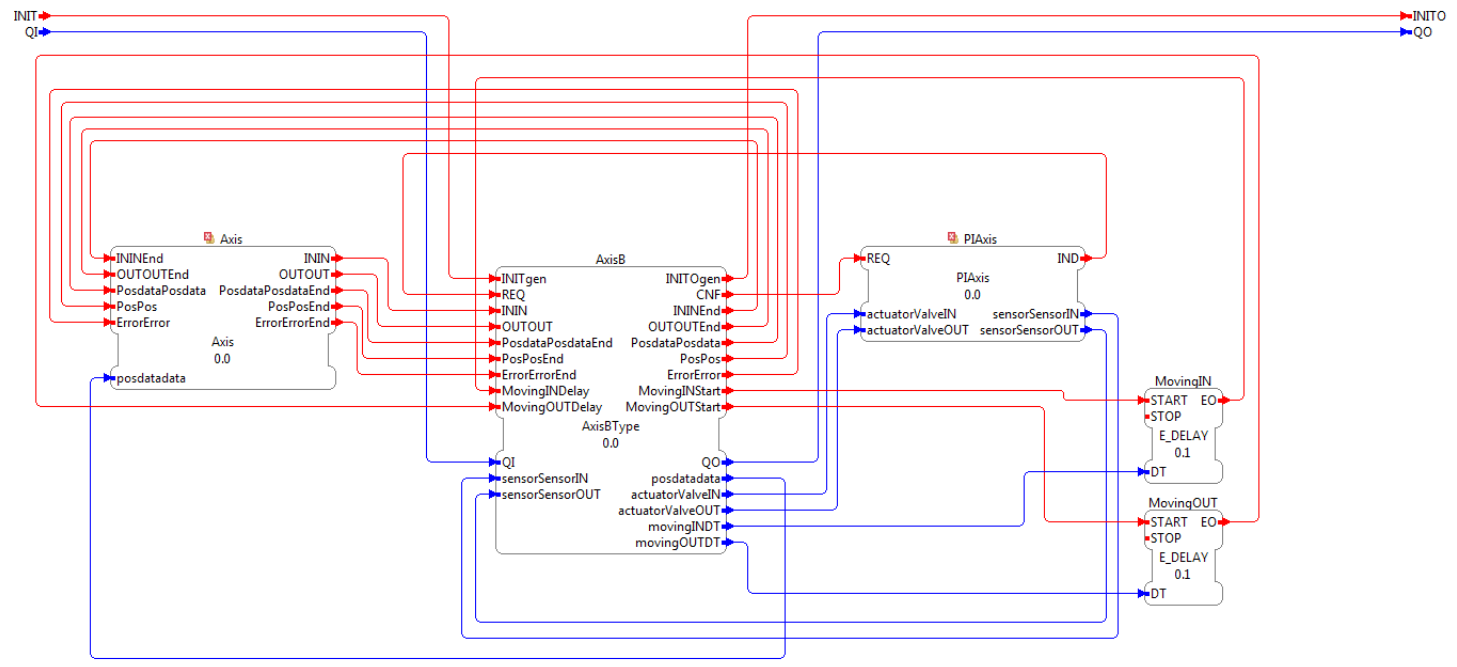
(a)



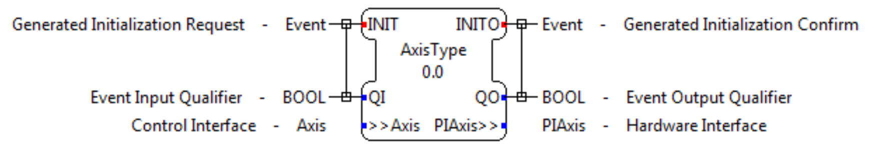
Generated Initialization Request - Event
Request - Event
CustomPort - comment - Event
CustomPort - comment - Event
CustomPort - comment - Event
CustomPort - comment - Event
CustomPort - comment - Event
Delay Event - Event
Delay Event - Event
Event Input Qualifier - BOOL
PlantPort - comment - BOOL
PlantPort - comment - BOOL
INITgen
REQ
ININ
OUTOUT
PosdataPosdataEnd
PosPosEnd
ErrorErrorEnd
MovingINDelay
MovingOUTDelay
QI
sensorSensorIN
sensorSensorOUT
INITOgen
CNF
ININEnd
OUTOUTEnd
PosdataPosdata
PosPos
ErrorError
MovingINStart
MovingOUTStart
AxisBType
0.0
QO
posdatadata
actuatorValveIN
actuatorValveOUT
movingINDT
movingOUTDT
Event - Generated Initialization Confirm
Event - Confirm
Event - CustomPort - comment
Event - CustomPort - comment
Event - CustomPort - comment
Event - CustomPort - comment
Event - CustomPort - comment
Event - Delay Event
Event - Delay Event
BOOL - Event Output Qualifier
INT - CustomPort - comment
BOOL - PlantPort - comment
BOOL - PlantPort - comment
TIME - Delay Time
TIME - Delay Time

(b)

Figure 5.8: Implementation of the Basic AC Axis: (a) Based on the specification of the timed state chart, the ECC of a Basic FB is generated. (b) The automatically generated interface of this Basic FB has interface elements (e.g. MovingINStart, MovingINDT, and MovingINDelay) to allow bringing time information in the transitions of the ECC.



(a)


Generated Initialization Request - Event
Event Input Qualifier - BOOL
Control Interface - Axis
INIT
QI
>>Axis
AxisType
0.0
INITO
QO
PIAxis>>
Event - Generated Initialization Confirm
BOOL - Event Output Qualifier
PIAxis - Hardware Interface

(b)

Figure 5.9: Implementation of the Basic AC Axis: (a) Timing information from the specification in the timed statechart is added as E_DELAY FBs. These and the according data and event connections are automatically included in the FBN of the Composite FB. Interface elements Axis (towards the higher hierarchical level) and PIAxis (plant interface) are also connected automatically to the Basic FB. (b) The interface of the Composite FB that represents the Basic AC Axis.

Composite Automation Component Composite ACs are also implemented as CFBs for portability reasons[8]. The behaviour implementation is reduced to the interconnection of interface elements of sub-components and provision of a stable interface for higher level ACs. Management functionality and more complex behaviour is modelled in a separate sub-AC (see Section 3.2). Therefore, the CFB's internal Function Block Network (FBN) only contains the plugs for the interaction with lower hierarchical levels and a socket, which is the interface towards higher levels.

The available interface elements are connected as specified in the AC's behaviour specification. All data and event interface elements provided by the adapters are accessible individually.

Hence, only a single CFB is generated for the implementation of a Composite AC and added to the IEC 61499 modelling element library.

5.2.4 Generation of Automatic Control and Simulation Application

The last step in the generation is the creation of an automatic control application and the related simulation application. The hierarchical structure of ACs is the basis for the generation of the automatic control application. FBs are instantiated accordingly and their adapter interfaces are automatically connected by the application generation mechanism as specified.

As the (fully) automatic application generation is mainly targeted at the simulation, the hardware access FBs are based on local multicast communication FBs. The CFBs contain PUBL- and SUBL-FBs. Hence, the automatic control application can be coupled with the related simulation application. Only the hardware access FB and the communication FB within the simulation application need to be deployed to the same automatic control device.

However, it is also feasible to automatically provide hardware access FBs, which provide the real process interfaces. Data from the ESM and the MM can be used as additional inputs for the model transformation process. Since the ICPs provide stable interfaces, the hardware access FBs can be replaced (dependent on the target control device) without need for further changes in higher levels.

The simulation application is also generated automatically. It has the same hierarchical structure as the related automatic control application. For the interaction with the automatic control application, an FB with the "mirrored" functionality of the previously described hardware access FB with PUBL- and SUBL-FBs is generated. The local multicast channels are connected by using the same unique IDs in both applications (e.g. for the same sensor or actuator).

[8]4DIAC-IDE and FORTE are currently the only environment that supports Adapters in BFBs.

The FBs generated for the plant simulation also feature adapters for the modelling of physical interfaces. These have to be connected according to the specification in the PlaM. Currently kinematic chains and material flows are the only supported physical interactions.

5.3 Simulation Execution System

In Section 4.5 the simulation of the plant behaviour by means of coupled automatic control applications is proposed. An event-based runtime environment for industrial automation, like the open-source runtime environment FORTE, is sufficient for the simulation of plant behaviour in nominal time. However, certain simulation scenarios require modifications of the used runtime environment. Furthermore, co-simulation setups (i.e. the integration of external simulation tools) are important for additional insight on the behaviour of the automated systems. Changes, both in the runtime environment but also the external tools might be necessary. The feasibility is shown with the integration of two different simulation environments: a network simulator and a continuous simulation environment.

5.3.1 Adaptation of Runtime Environment FORTE for Simulation

Time Advancement

An important feature of simulation environments is a time advancement that is independent of the nominal time [66, 67]. Both increased and decreased time advancement have their justification, dependent on the simulation scenario. Two different implementations for the time advancement have been provided: fixed rate of the nominal time (e.g. half or double advancement rate compared to nominal time) and variable time advancement rate [111].

Fixed Rate A fixed rate acceleration or deceleration of the time advancement, is reached by modifying the platform specific timer handler of the runtime environment FORTE. Within a special thread the nominal time is read from the operation system or hardware. After a specified amount of time has elapsed, the timer handler is notified. Hence, the modification of the time, which has to elapse before the notification is sent, results in the desired acceleration or deceleration. To prevent overload of the runtime environment (e.g. event-chains that cannot accept additional event-entries) in accelerated mode, the acceleration factor has to harmonize with the utilization (e.g. size of the automatic control application). Calculations of the Worst Case Execution Time (WCET) can be applied for the automatic control application and the simulation application. Zoitl shows that keeping the real-time execution constraints can be guaranteed as long as the synthetic utilization is less than 58.6% [62].

Decelerated operation should not face that problem, since the runtime environment has more time to execute than in normal operation mode. This mode may be necessary if a co-simulation setup with an external simulation tools, which runs at decelerated time advancement, is desired. If the external tool and FORTE run both at exactly the same time advancement rate, clock synchronisation can be neglected. Hence, built-in functionalities are used for setting the time adavancement rate and modifications of the commercial simulation environments (e.g. clock management functionality) are not necessary. Communication and data exchange between the involved simulation environments is done in an event-based manner. For hybrid simulation scenarios all simulators have to run at nominal time, like the involved operational parts of the plant and automatic control system.

Variable Rate A variable time advancement rate is possible, if the whole application (i.e. simulation and automatic control application) is deployed to a single device, or all devices support time synchronsation. The timer-handler can take the functionality of the Future Event List. Both the simulated plant as well as the automatic control application are reactive systems. Within the system boundaries any behaviour is modelled by timed statecharts. In that case external event sources other than the timer handler can be neglected.

As soon as no FB in the simulation or automatic control application is active, and no further FBs are to be activated due to events in the event chains, the timer-handler advances the clock of the runtime to the due time of the next timer event (i.e. timed FB). Hence, waiting time between timer events is skipped and accelerated time advancement is reached.

The synchronisation of multiple simulation devices running FORTE requires the exchange of the next entry in the timer-handler of any involved runtime environment. As soon as a runtime environment goes idle, the next entry is sent to a multicast channel, which all runtime environments have subscribed.

Simulation Resource

The implementation of a special Simulation Resource (see Section 4.5.4), which is based on the standard EMB_RES, enables a better transistion between simulation and operation and vice versa. It features a modified object handler. The object handler is responsible for the management (e.g. instantiation, deletion) of IEC 61499 modelling elements (e.g. FBs, connections) within an execution entity.

For an FB that needs to be changed for simulation (e.g. a hardware access FB) a related FB-type has to be provided, for example *FB1* and *SIM_FB1*. During the instantiation of new FBs the prefix "SIM_" is added automatically to the FB-type by the object handler. If a simulation specific FB-type is present, it is instantiated. Otherwise the failed instantiation is repeated without the prefix, and the general FB-type (i.e. hardware independent FB) is instantiated. This is an implementation of the factory method design pattern that is known in software engineering [198].

Hence, no functional changes to the automatic control application are necessary in the engineering tool. The change of the mapping is sufficient to switch between simulation and operation mode.

5.3.2 Network Simulation

The influence of network-properties is relevant in case of decentralized and distributed execution of automatic control applications, as multiple automatic control devices or I/O interfaces have to communicate via fieldbus systems or (industrial) networks. The inclusion of a specialised network simulator helps to estimate and validate the utilisation of these networks, and the influence of delays or jitter to the behaviour of networked automatic control applications. Hence, this analysis can be brought from the commissioning phase, where it currently is done [183], to the implemenation phase.

For this purpose the open source network simulator ns3 has been included in the simulation execution system [186]. It has been developed mainly by academic users and is actively maintained and developed. Its modular structure and rich libraries allow the creation of comprehensive and complex network models. Furthermore, it is easily possible to modify modules dependent on the needs of the simulation scenario.

On basis of the ns3 network simulator, the ns3-DCE (Direct Code Execution Environment) has been provided by researchers of INRIA[9] [200, 201]. An abstraction layer is introduced, which redirects all network related system calls towards the network simulation. However, the presented project is only operational on specific Linux distributions, since dependencies with the system libraries of Linux occur.

Nevertheless, it is a promissing approach to introduce real applications, which use network communication, into network simulation. With only minor modifications to the source code of FORTE it was possible to include the runtime environment into a network simulation scenario. Multiple instances of the IEC 61499 runtime environment operate in a simulated network envi-

[9]The French Institute for Research in Computer Science and Automation (Institut national de recherche en informatique et en automatique, INRIA) is a founding member of the NS3 consortium [186, 199].

ronment as virtual automatic control devices. Hence, effects of the distribution can be evaluated. The simulation parameters of the virtual network that is coupling the virtual automatic control devices can be easily adjusted. So it is possible to investigate the influence of the network onto the behaviour of the automatic control system. The computer running the network simulation is acting as gateway device, as can be seen in Figure 5.10. This functionality is needed at least for the deployment of the automatic control applications, as Java-based tools currently cannot be executed within the ns3-DCE environment [201]. However, it is even possible to combine the simulated network with a real network, where other automatic control devices are attached to. Hence, the network simulation can also be used in full as well as hybrid simulation scenarios.

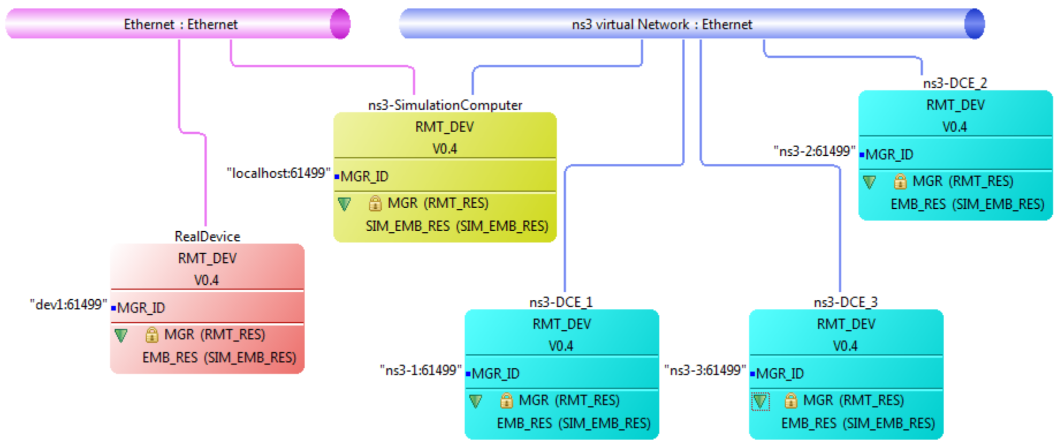


Figure 5.10: Network simulation scenario with 3 virtual ns3-DCE devices, the gateway device (i.e. simulation computer), and a real automatic control device. Via the gateway device automatic control applications can be deployed to virtual as well as to real automatic control devices. The effect of the network and mapping decisions can be evaluated in depth, as the network parameters (e.g. packet loss, network utilization, bandwith) can be easily adjusted to resemble realistic or even worst case behaviour.

5.3.3 Integration of Dymola

Dymola [202] is a powerful tool for the simulation of continuous processes. Hence, many reliable models and modelling libraries exist and are widely accepted. To make this models accessible in automatic control development, but also to provide a realistic and reliable behaviour of automatic control to process engineers, Dymola has been included in the simulation environment. Therefore, Dymola and FORTE have to be connected to being able to exchange events and data. Furthermore, the time of both simulation entities has to be kept synchronous.

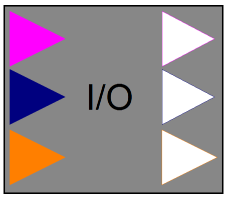
Connecting FORTE and Dymola

Network based communication is used to couple the simulation execution of Dymola with FORTE. To reduce the communication effort on the one hand and to allow to adjust the number of exchanged data points on the other hand the link between the two programs is established via a string-based protocol. On the Dymola side the network communication is provided by a static library programmed with Visual C++, which is linked to the simulation execution program. As there does not exist a standard encoding in Dymola, the data encoding specified in IEC 61499-1 [60] and the IEC 61499 Compliance Profile for Feasibility Demonstrations [203] is applied. Hence, the TCP Client within Dymola can be directly connected to an unmodified SERVER_1_1 communication FB. One STRING datatype in each direction is sufficient to send and receive the encoded data.

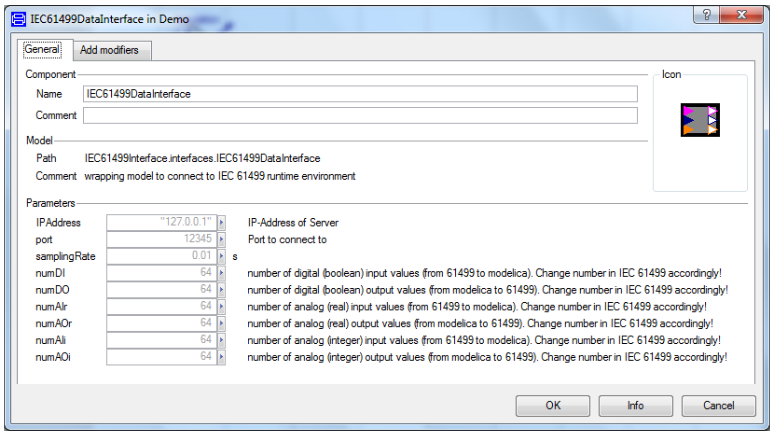
The elementary datatypes in Dymola are: Boolean, 32-bit Integer, and 64-bit Float. These are equivalent to the IEC 61131 datatypes BOOL, DINT, and LREAL. The Dymola modeling element *IEC61499DataInterface*, which is shown in Figure 5.11(a), provides a configureable number of Boolean, Integer and Real inputs at the left side, and outputs on the right side (see Figure 5.11(b)). Values from Dymola to IEC61499 and vice versa are transformed to a String and concatenised. On the receiving end, the string is split into 3 Strings (each for a datatype) and further evaluated. Finaly the values are provided as single value, which can be directly connected to the according model (IEC61499 or Dymola).

Time Synchronisation

Dymola provides interfaces to synchronise the time with external tools. However, the time synchronisation mechanism of Dymola cannot be generalized for other tools. Therefore, a fixed ratio of time advancement is used. The external model JPAARealTime [204] provides an interface to include such a fixed time advancement in Dymola (see Figure 5.12). Its main purpose is to ensure realtime execution of the simulation model, but also other factors than 1 can be used. Hence, increased or decreased time advancement can be reached. The time advancement of the coupled FORTE runtime environment just has to be set to the same value. The JPAARealTime model also provides a diagram, which shows the deviation from the desired time advancement. If the execution of the simulation model takes longer than desired then time deviations increase during the simulation run. The JPAARealTime model is only able to modify the virtual simulation time when the simulator is in idle mode. Hence,



(a)



IEC61499DataInterface in Demo

Component

Name	IEC61499DataInterface
Comment	

Model

Path	IEC61499Interface.interfaces.IEC61499DataInterface
Comment	wrapping model to connect to IEC 61499 runtime environment

Parameters

IPAddress	"127.0.0.1"		IP-Address of Server
port	12345		Port to connect to
samplingRate	0.01	s	
numDI	64		number of digital (boolean) input values (from 61499 to modelica). Change number in IEC 61499 accordingly!
numDO	64		number of digital (boolean) output values (from modelica to 61499). Change number in IEC 61499 accordingly!
numAIr	64		number of analog (real) input values (from 61499 to modelica). Change number in IEC 61499 accordingly!
numAOr	64		number of analog (real) output values (from modelica to 61499). Change number in IEC 61499 accordingly!
numAIi	64		number of analog (integer) input values (from 61499 to modelica). Change number in IEC 61499 accordingly!
numAOi	64		number of analog (integer) output values (from modelica to 61499). Change number in IEC 61499 accordingly!

(b)

Figure 5.11: Configureable Dymola modelling element for data exchange with IEC 61499 automatic control applications. (a) Representation in Dymola simulation model. (b) Configurable parameters of modelling element.

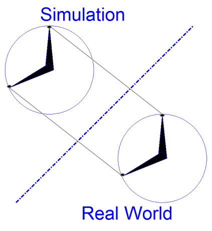
the solver or model can be changed that less time is needed for the calculation of the model. However, this also influences the results of the simulation run. Another possibility to keep the simulation clocks syncronous is to chose another time advancement rate.

5.4 Evaluation Test Cases

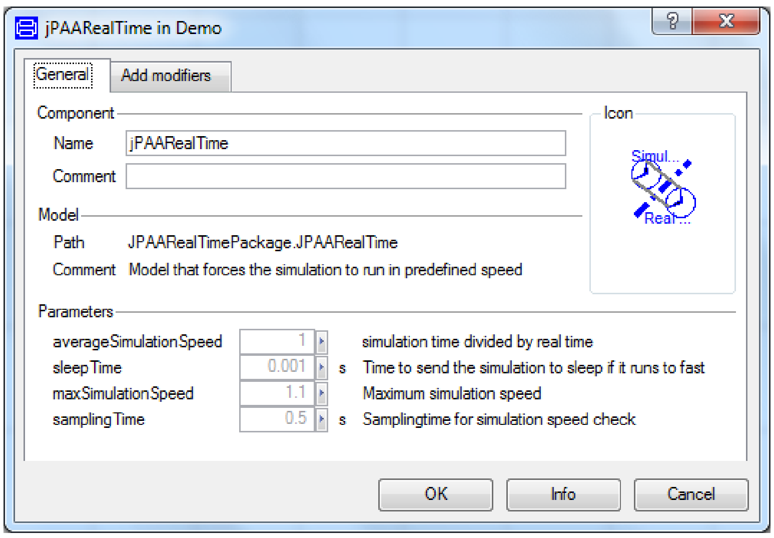
The proposed engineering and simulation approach is mainly focused on the automation of (discrete) manufacturing systems. However, also the automation of (continuous) process engineering plants shall be feasible. Test cases from three domains—discrete manufacturing plants, robotic applications, and process engineering—are used to evaluate the feasibility ot the developed concept. Each domain specific test case covers special aspects and requirements, which have to be fulfilled by the modelling and simulation workflow.

5.4.1 Discrete Manufacturing Plants

Discrete manufacturing plants will have to adopt to changing requirements [21]. Flexibility to produce various goods—with lot-size one production in focus—will result in ever-changing manufacturing systems. Hence, the validation of changed setups by means of simulation is a promising approach to reduce downtimes during reconfiguration of the plants.



(a)



Component

Name	jPAARealTime
Comment	

Model

Path	JPAARealTimePackage.JPAARealTime
Comment	Model that forces the simulation to run in predefined speed

Parameters

averageSimulationSpeed	1		simulation time divided by real time
sleepTime	0.001	s	Time to send the simulation to sleep if it runs to fast
maxSimulationSpeed	1.1		Maximum simulation speed
samplingTime	0.5	s	Samplingtime for simulation speed check

(b)

Figure 5.12: Configureable Dymola modelling element for time synchronisation with realtime clock. (a) Representation in Dymola simulation model. (b) Configurable parameters of modelling element.

Test Case Description

A representative example within this domain is the part sorting machine. The part sorting machine represents a single working cell and is presented in Figure 5.13. This machine, which is available at ACIN, can be operated as stand-alone application or integrated in a more complex manufacturing system with a palette transfer system. The part sorting machine features:

- Part identification: Parts, which are transported on palettes, can be classified by either a colour sensor or an inductive sensor.
- Part handling: Two identical 2-DOF handling units, each consisting of 2 pneumatic axes and a pneumatic gripper, are able to move parts from the transportation palette to a part depot.
- Internal transportation: Within the machine a conveyor belt provides transportation services for the palettes. To allow the identification and handling of parts stoppers are used for the fixation of the palettes.

With the selected discrete manufacturing machine the following aspects of the design and simulation approach can be validated:

- Control behaviour modelling process: Starting from discipline specific engineering tools, the control behaviour of the plant is specified. This test

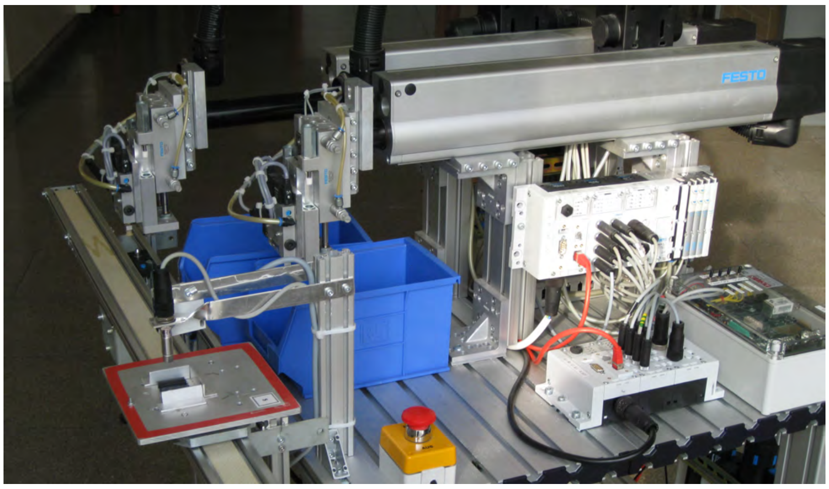


Figure 5.13: Laboratory application "Part Sorting Machine" is used as example for a discrete manufacturing process.

case allows to evaluate the feasibility of reuse or multiple instantiation of similar components (e.g. axes, handling units).

- Control application generation: The control application is to be automatically generated from the ACIM for the specified and available control platform.
- Plant behaviour modelling: The applicability of the proposed simulation approach—from behaviour modelling to the generation of the plant simulation application—can be evaluated with the provided test case. Full simulation as well as hybrid simulation (e.g. one handling unit simulated) scenarios can be applied.
- Material handling: The modelling as well as the implementation of material flows in simulated plants can be tested with the intra-process logistic process in the part sorting machine.
- Simulation application generation: The simulation application which represents the plant behaviour (including material handling) shall be automatically generated as IEC 61499 application that is linked to the related control application.

Experiments and Evaluation

The mechanical structure of the test case was already available. Nonetheless, the machine has been logically split into self-contained entities in a top-down approach. The resulting module structure is presented in Figure 5.14.

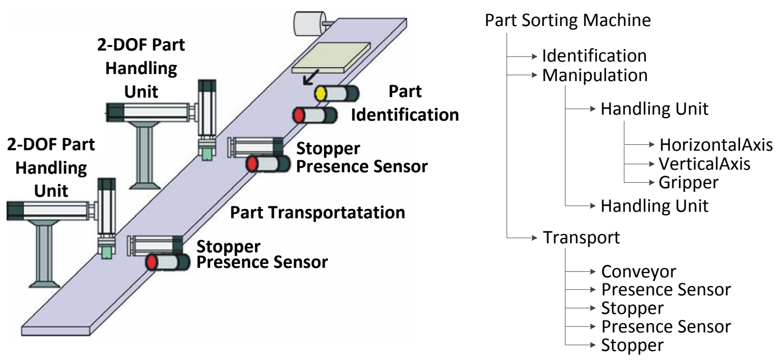


Figure 5.14: Schematic and hierarchical structure of the laboratory application “Part Sorting Machine”.

The structure, the interfaces, and the behaviour of the identified components have been modelled with UML diagrams with StarUML. The UML Class Diagram that specifies the interface of the AC *HandlingUnit* and generically defines the contained sub-components is presented in Figure 5.15. The concrete setup for a specific AC instance is provided in form of a UML Object Diagram. This diagram type allows to include multiple sub-components of the same type and model their interaction (see Figure 5.17). The behaviour of ACs is modelled with UML Statechart Diagrams. The behaviour specification for the AC *Axis* is presented in Figure 5.16. This AC is a Basic AC, with hardware interaction, and a sub-component of the *HandlingUnit*. The same modelling methodology has been applied for the specification of both the automatic control and the plant simulation applications.

This fact has been deemed useful by the users. However, the same users were responsible for modelling both aspects in this test case and all comparable test cases for discrete manufacturing plants. On the one hand, the UML profile acts as limitation in the expressiveness and requires additional modelling effort (e.g. additional windows to set properties). On the other hand, these restrictions enabled a highly automated information extraction process with little to no need for manual changes in the resulting ACM.

All modelling tasks for this test case are done from scratch. Due to the additional structuring tasks, which increase the reusability, the initial effort is higher than the direct automatic control application programming would be. As a result of the proposed engineering approach, components with well defined behaviour and interfaces are created. Industrially applied automatic control applications require well documented process interfaces anyway. The automatically generated simulation application, which allows to test the implementation of the automatic control application early in the development cycle, can be reached with little additional effort. Especially information on physical connections have to be specified explicitly in the proposed approach.

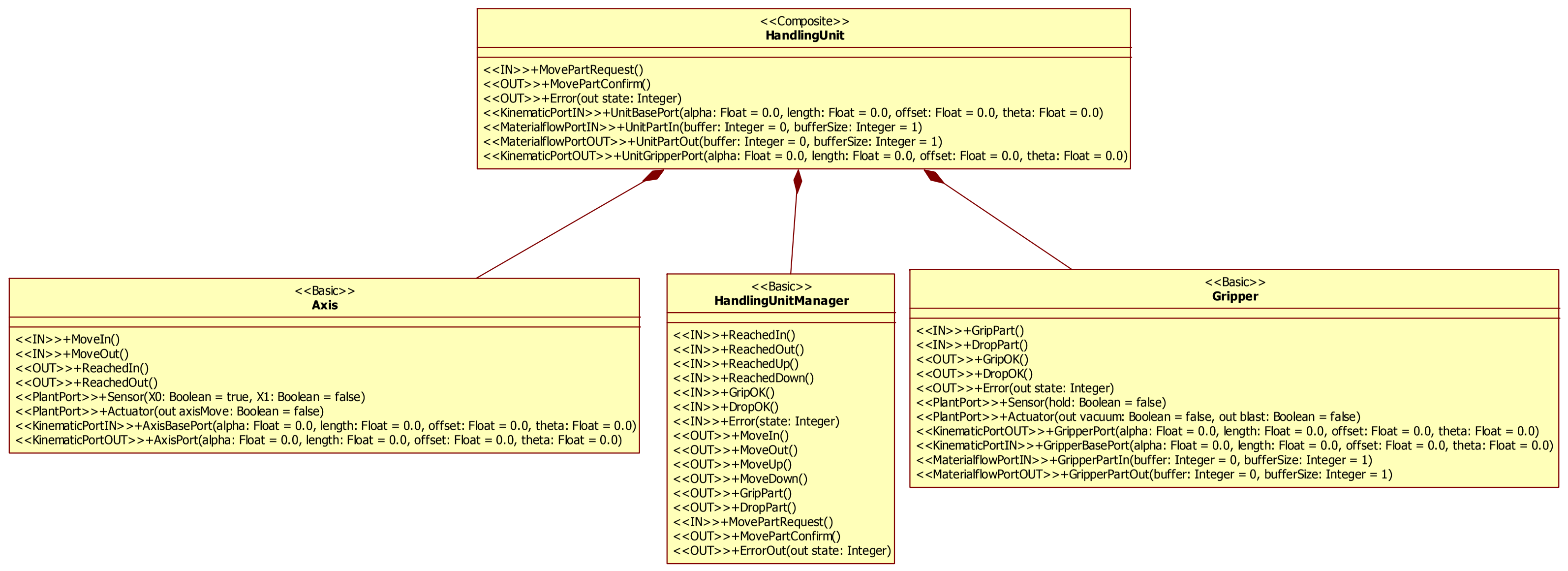

<<Composite>>
HandlingUnit
<<IN>>+MovePartRequest()
<<OUT>>+MovePartConfirm()
<<OUT>>+Error(out state: Integer)
<<KinematicPortIN>>+UnitBasePort(alpha: Float = 0.0, length: Float = 0.0, offset: Float = 0.0, theta: Float = 0.0)
<<MaterialflowPortIN>>+UnitPartIn(buffer: Integer = 0, bufferSize: Integer = 1)
<<MaterialflowPortOUT>>+UnitPartOut(buffer: Integer = 0, bufferSize: Integer = 1)
<<KinematicPortOUT>>+UnitGripperPort(alpha: Float = 0.0, length: Float = 0.0, offset: Float = 0.0, theta: Float = 0.0)
<<Basic>>
Axis
<<IN>>+MoveIn()
<<IN>>+MoveOut()
<<OUT>>+ReachedIn()
<<OUT>>+ReachedOut()
<<PlantPort>>+Sensor(X0: Boolean = true, X1: Boolean = false)
<<PlantPort>>+Actuator(out axisMove: Boolean = false)
<<KinematicPortIN>>+AxisBasePort(alpha: Float = 0.0, length: Float = 0.0, offset: Float = 0.0, theta: Float = 0.0)
<<KinematicPortOUT>>+AxisPort(alpha: Float = 0.0, length: Float = 0.0, offset: Float = 0.0, theta: Float = 0.0)
<<Basic>>
HandlingUnitManager
<<IN>>+ReachedIn()
<<IN>>+ReachedOut()
<<IN>>+ReachedUp()
<<IN>>+ReachedDown()
<<IN>>+GripOK()
<<IN>>+DropOK()
<<IN>>+Error(state: Integer)
<<OUT>>+MoveIn()
<<OUT>>+MoveOut()
<<OUT>>+MoveUp()
<<OUT>>+MoveDown()
<<OUT>>+GripPart()
<<OUT>>+DropPart()
<<IN>>+MovePartRequest()
<<OUT>>+MovePartConfirm()
<<OUT>>+ErrorOut(out state: Integer)
<<Basic>>
Gripper
<<IN>>+GripPart()
<<IN>>+DropPart()
<<OUT>>+GripOK()
<<OUT>>+DropOK()
<<OUT>>+Error(out state: Integer)
<<PlantPort>>+Sensor(hold: Boolean = false)
<<PlantPort>>+Actuator(out vacuum: Boolean = false, out blast: Boolean = false)
<<KinematicPortOUT>>+GripperPort(alpha: Float = 0.0, length: Float = 0.0, offset: Float = 0.0, theta: Float = 0.0)
<<KinematicPortIN>>+GripperBasePort(alpha: Float = 0.0, length: Float = 0.0, offset: Float = 0.0, theta: Float = 0.0)
<<MaterialflowPortIN>>+GripperPartIn(buffer: Integer = 0, bufferSize: Integer = 1)
<<MaterialflowPortOUT>>+GripperPartOut(buffer: Integer = 0, bufferSize: Integer = 1)

Figure 5.15: UML Class Model for Automation Component "Handling Unit".

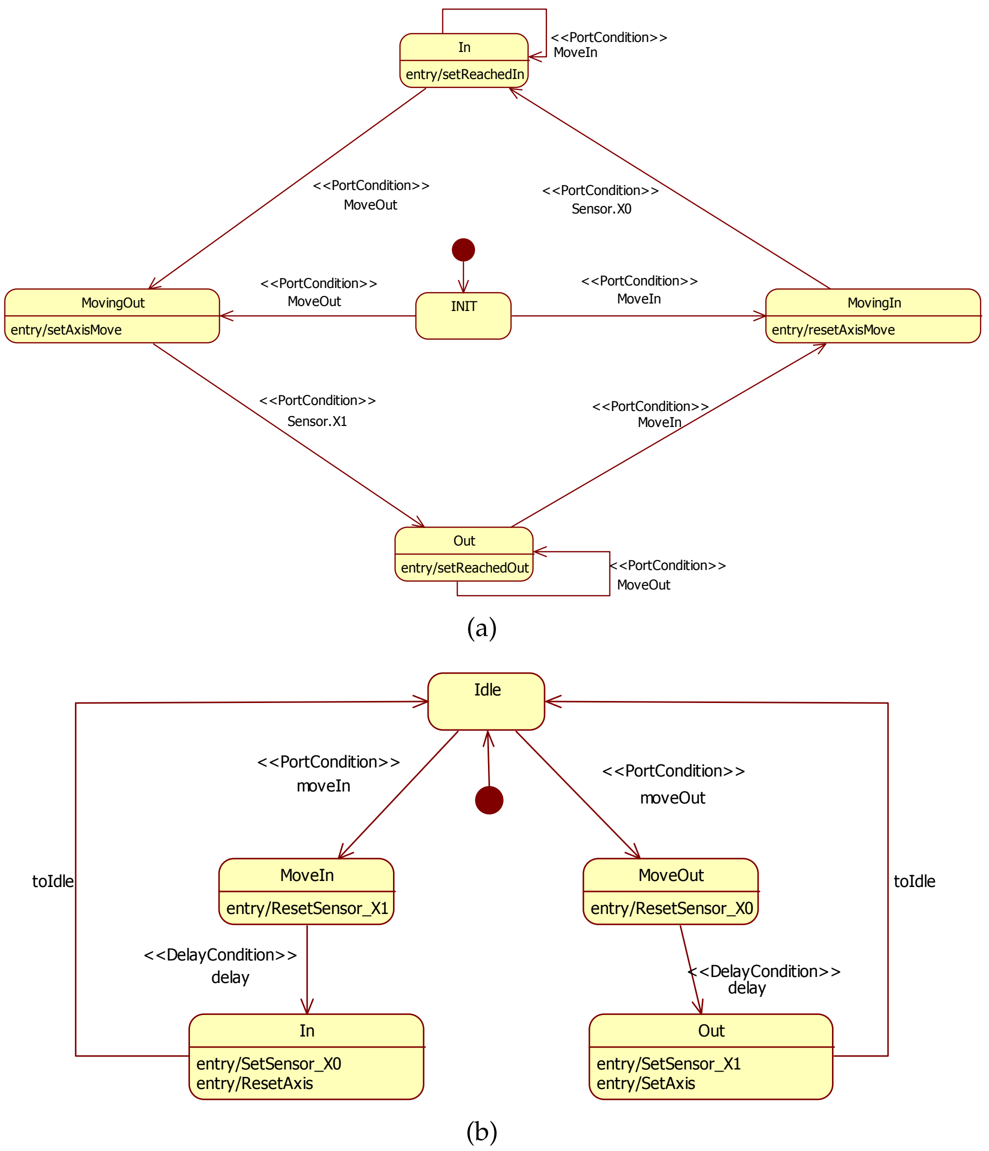


Figure 5.16: Behaviour Model in form of a UML Statechart Diagram for the Automation Component Axis. (a) Specification of the automatic control behaviour of the generic AC Axis without error handling functionalities. (b) Specification of the plant behaviour for the AC Axis, which is used for the simulation.

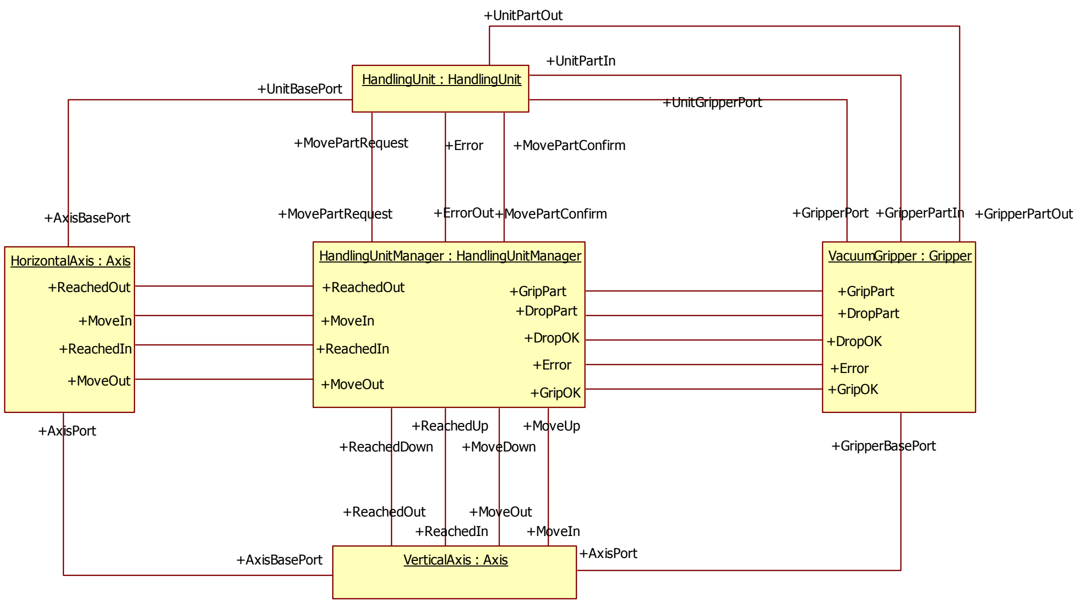


Figure 5.17: UML Object Model for Automation Component "Handling Unit", composed of 2 Axes and 1 Gripper.

These are often inaccessible or well hidden from automatic control designers. Based on the explicit specification material flow is automatically added to the plant simulation application. In the test case material is provided to one input and can leave the machine at three different outputs.

The full simulation scenario, including material flow simulation, has been initially performed on a PC with the adapted FORTE runtime environment. Changes of the target platform are easily possible with the proposed model-based engineering and modelling framework. Later, the full simulation scenario has been executed on the target execution system, consisting of 3 valve terminals from FESTO. Finally, the automatically generated automatic control application (see Figure 5.18) has been deployed to the automatic control system of the plant in the laboratory. The same functionality as with the manually programmed automatic control application has been observed. The generic automatic control application generation results in slightly more overhead compared to applications manually programmed specific for an application. However, the generated applications (both automatic control as well as simulation) are well structured and readible.

This test case, although only covering a single machine, already revealed the benefit of component reuse. Within a single handling unit two axes are used. Although they differ mechanically, the same component type has been used as basic model. They both offer the same interface to the higher hierarchical levels. Only the interface to the plant has to be modified. Both the timing of the plant and the I/O configuration change.

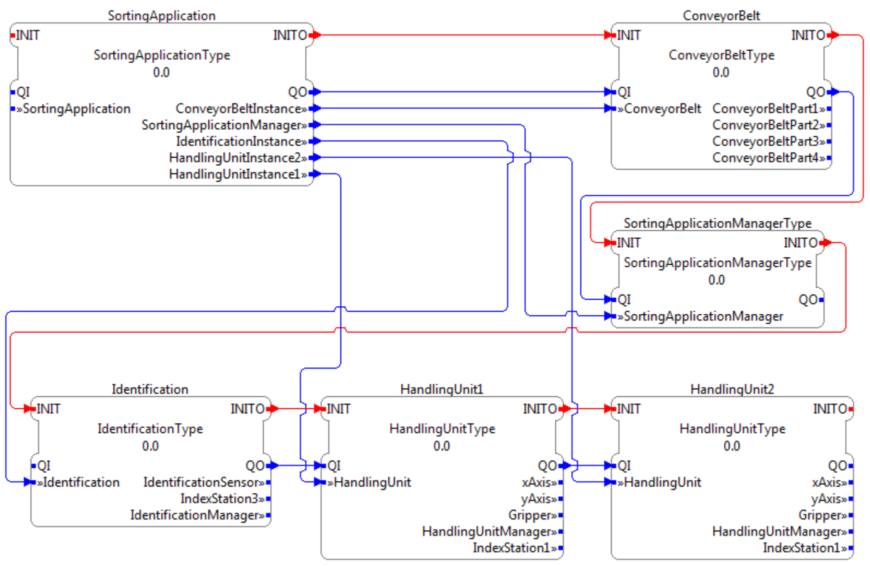
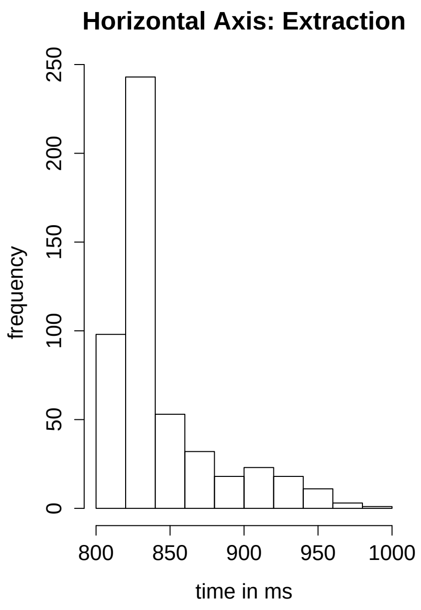


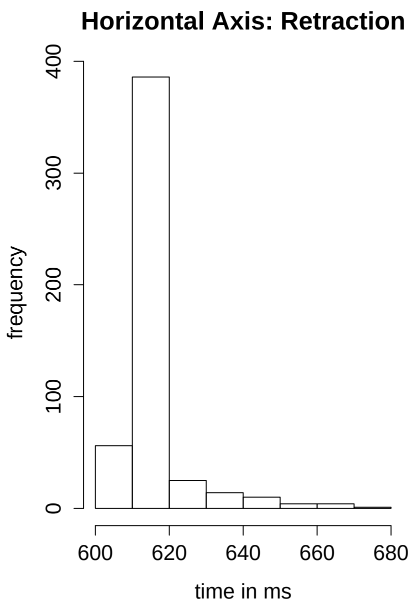
Figure 5.18: Part of the automatically generated, hierarchical automatic control application for the "Part Sorting Machine".

For the validation of the behaviour simulation for the horizontal axis, with a special focus on the extraction and retraction times, simulation assumptions and results are compared to experimental results. Under laboratory conditions, that means constant pressure of 4 bar and room temperature, a sample of 500 cycles is drawn. The histograms of the measured times is shown in Figure 5.19(a) and Figure 5.19(b) respectively. According to the results of these measurements a large number of samples show a high degree of repeatability for extraction (e.g. 76,6% within a range of 5 ms for extraction of the axis) whereas the variance and also absolute times are higher for the retraction. The reason for the different behaviour for extraction and retraction are probably related to the construction of the axis. However, a further analysis for the reasons of the axis' behaviour is not in the scope of this thesis.

In the sample of 500 measured cycles all time values (extraction and retraction) lie below 1000 ms, which has been used as time during the simulation. Hence, the chosen time frame clearly represents a worst case scenario, and the gained results are representative under the given conditions. However, another sample of 100 cycles has been drawn under different conditions (pressure of 3.8 bar and temperature of approx. 5 degrees Celsius). The first cycles under these conditions show major deviations from laboratory conditions. During ramp up of the axis the first 20 cycles require up to 1900 ms. These deviations seem to stem from viscous lubrication and viscous oil in the dampers that dampen the movement at both end positions. However, after the warm up phase all further measured times drop significantly. For this reason, simulation results for the movement of the axis are reasonable for sufficiently



(a)



(b)

Figure 5.19: Histogram of measured times for 500 cylcles of extraction and retraction of the horizontal axis under laboratory conditions.

high number of cycles or for steady operation.

The efficiency in the development with the proposed approach is expected to increase and outperform the currently used development approaches as soon as a comprehensive library of components has been created.

5.4.2 Robotic Applications

Robotic and kinematic applications are also important in flexible manufacturing systems. Both discrete pick-and-place operations and continuous path-following operations are in the focus. Robotic components are commonly mechatronic components, which include the physical component (mechanics and electrics) but also control functionalities and well-defined interfaces.

Test Case Description

Representative examples for robotic components are provided by Schunk. Their Powercubes and Powerballs provide 1 or 2-DOF movement. Each component is self-contained and provides a control interface via CAN fieldbus. Larger components can be created by putting together multiple of such elementary robotic components. For the evaluation of the proposed engineering

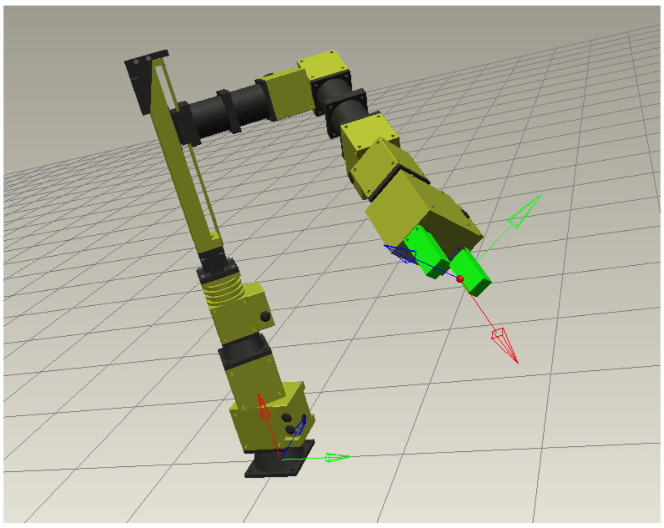


Figure 5.20: Robotic Application built from multiple elementary robotic components from Schunk. The test setup is depicted in Schunk's configuration tool for the creation and validation of robotic applications ViroCon.

and simulation approach a test setup built from multiple elementary robotic components (i.e. 3 rotating units, 1 linear unit, 1 gripper) has been available as depicted in Figure 5.20.

The presented robotic test application is able to cover a set of important engineering and simulation aspects, that go further than the previous test case:

- Hierarchical aggregation of components: Existing components are used to create the desired robotic application.
- The lower level components cannot be altered. commercial-of-the-shelf (COTS) components (e.g. provided by different vendors) are integrated by using their services. A fieldbus system (i.e. CAN) is used to provide the interfaces to the integrating level. Hence, the interface between simulation and control is also put at this abstraction level.
- The simulation of kinematic chains, using Denavit-Hartenberg parameters, can be well evaluated with this module-based robotic application.

Experiments and Evaluation

The configuration tool for robotic applications ViroCon is used as domain specific tool. Based on the information provided by this tool—a list of robotic components (see Listing 5.3 and component movements—an AC is created.

The AC includes a sub-AC for the manager functionality, which specifies the coordinated movements of all sub-components. The other sub-components represent basic robotic components provided by Schunk.

Listing 5.3: XML-based listing of robotic components provided by ViroCon.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
-<module_list>
+<module name="PAM112">
-<module name="PR90">
  <parameter name="ID" value="PR90"/>
  <parameter name="max" value="175"/>
  <parameter name="min" value="-175"/>
 </module>
+<module name="PAM110">
{...}
```

For each basic robotic component type, an AC is provided. These Basic ACs do not have their own specific behaviour specification. Templates are provided by the ViroCon import functionality (see Figure 5.21 and Figure 5.22).

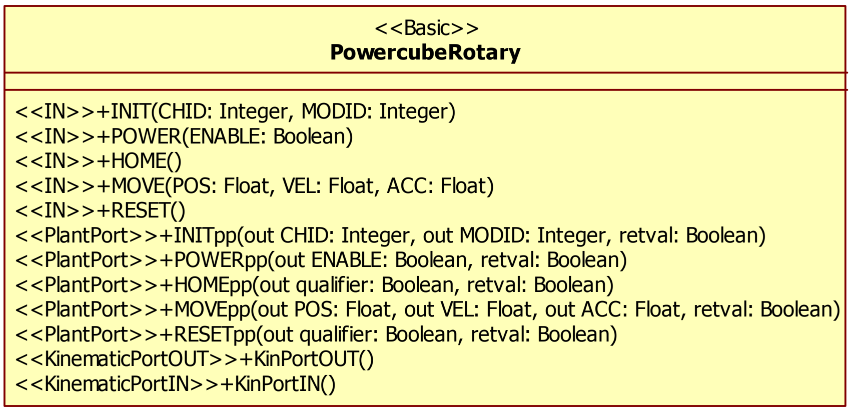
The Basic ACs only act as proxies for the components. Hence, the module behaviour specifications are only needed for the behaviour simulation. All robotic components of Schunk share the same interface towards the automatic control application, which cannot be altered.

The plant model further includes probabilistic timing information to provide realistic simulation results. Dependent on the movements of the robotic components the kinematic chain is recalculated. The correctness of the calculation of the position and orientation of the end-effector has been validated in multiple simulation runs, which have been compared to runs on the real plant.

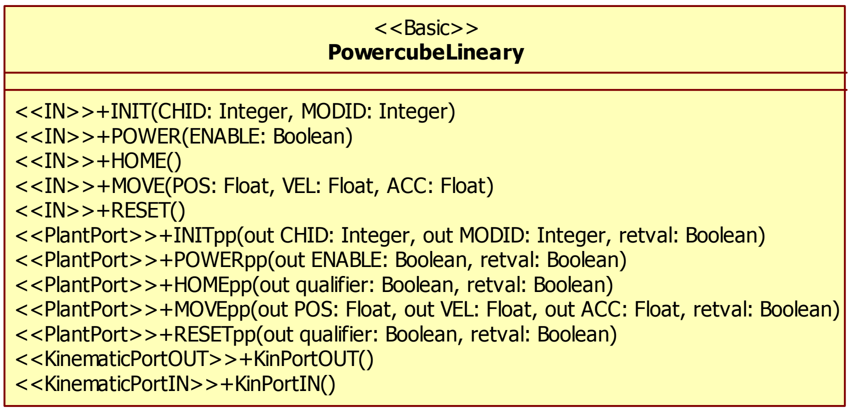
Also the transition from simulation to operation has been proven to be feasible. In this case, not only the I/O access FBs are replaced. The robotic components only act as proxies for the real components. Hence, also the lowest level in the automatic control application is not deployed.

5.4.3 Process Technology

Modernisation of automation equipment of continuous processes is an important application field for model-based engineering and simulation approaches. Such process plants include for example electrical power or petro-chemical plants. Due to possible hazardous results of failures in operating plants simulation is often used during the development process. The integration of available simulation models (e.g. for Dymola/Modelica) provides a reliable abstraction of the plant behaviour during the development of discrete automation applications.


<<Basic>>
PowercubeRotary
<<IN>>+INIT(CHID: Integer, MODID: Integer)
<<IN>>+POWER(ENABLE: Boolean)
<<IN>>+HOME()
<<IN>>+MOVE(POS: Float, VEL: Float, ACC: Float)
<<IN>>+RESET()
<<PlantPort>>+INITpp(out CHID: Integer, out MODID: Integer, retval: Boolean)
<<PlantPort>>+POWERpp(out ENABLE: Boolean, retval: Boolean)
<<PlantPort>>+HOMEpp(out qualifier: Boolean, retval: Boolean)
<<PlantPort>>+MOVEpp(out POS: Float, out VEL: Float, out ACC: Float, retval: Boolean)
<<PlantPort>>+RESETpp(out qualifier: Boolean, retval: Boolean)
<<KinematicPortOUT>>+KinPortOUT()
<<KinematicPortIN>>+KinPortIN()

(a)


<<Basic>>
PowercubeLineary
<<IN>>+INIT(CHID: Integer, MODID: Integer)
<<IN>>+POWER(ENABLE: Boolean)
<<IN>>+HOME()
<<IN>>+MOVE(POS: Float, VEL: Float, ACC: Float)
<<IN>>+RESET()
<<PlantPort>>+INITpp(out CHID: Integer, out MODID: Integer, retval: Boolean)
<<PlantPort>>+POWERpp(out ENABLE: Boolean, retval: Boolean)
<<PlantPort>>+HOMEpp(out qualifier: Boolean, retval: Boolean)
<<PlantPort>>+MOVEpp(out POS: Float, out VEL: Float, out ACC: Float, retval: Boolean)
<<PlantPort>>+RESETpp(out qualifier: Boolean, retval: Boolean)
<<KinematicPortOUT>>+KinPortOUT()
<<KinematicPortIN>>+KinPortIN()

(b)

Figure 5.21: Generic UML Class Diagrams for (a) rotary and (b) linear Power-Cube Modules.

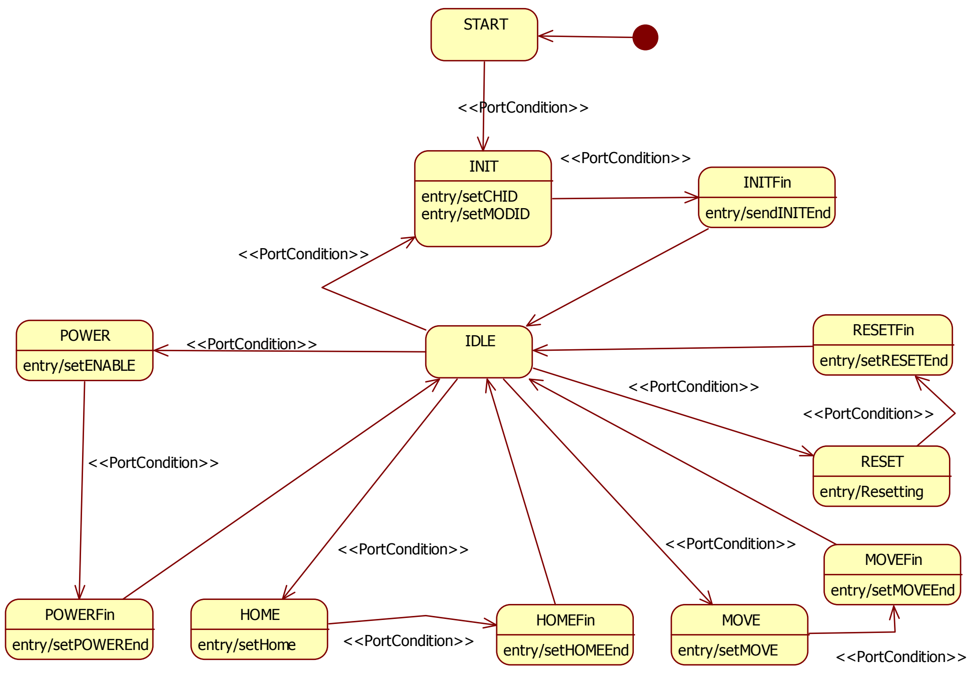
Test Case Description

For the evaluation of the engineering approach, a Dymola simulation model of the chilling circle of a thermal power plant has been available. An existing feedback control algorithm for the regulation of the temperature (regulating and switching valves are accessible) has to be transferred to a new control platform. The Dymola model has been provided by EDF[10] for the evaluation of the proposed approach. Due to IPR reasons the model is not included in this thesis.

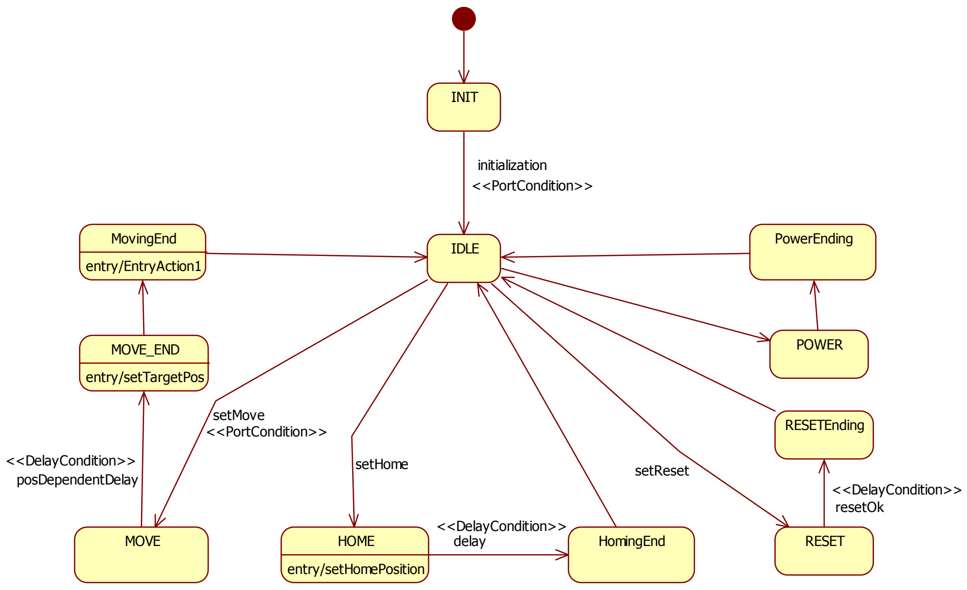
This test case allows to validate:

- Integration of an external simulation tool (i.e. Dymola) in the plant simulation application.

[10]The French company Électricité de France (http://www.edf.com) was an industrial partner in the MEDEIA project.



(a)



(b)

Figure 5.22: Generic UML Statechart Diagrams for PowerCube Modules: (a) Behaviour Model and (b) Plant Behaviour Model.

Experiments and Evaluation

The feedback control algorithm is already available as specified AC. The automatic control application is automatically generated with the available model transformation facilities. However, an automatic coupling of the legacy Dymola model and the automatic control application is not feasible. Therefore, the I/O access FB is manually implemented. Its internal FBN, which is shown in Figure 5.23 is encoding the values into a STRING and sends it to the Dymola application via a standard SERVER FB. STRING values received from Dymola are decoded and provided at the interface of the I/O access FB. The exchanged values represent sensor values, which are provided by the Dymola model, and actuator values, which are sent to Dymola.

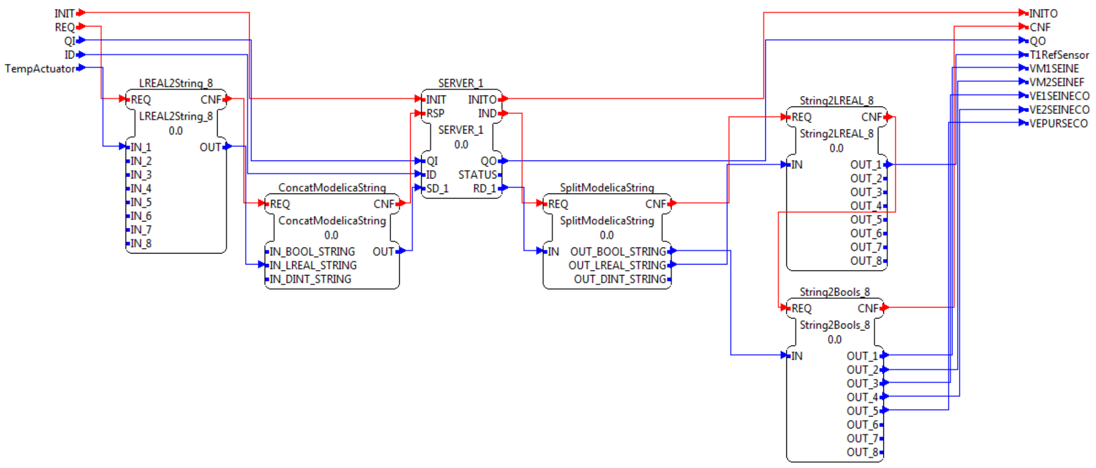


Figure 5.23: I/O access FB for coupling FORTE and Dymola.

As actuators in the primary circuit two manually actuated, binary valves (open/close) and a controlable proportional valve are available. The Dymola model provides various process values, like temperature and pressure values. But only values required in the feedback controller (specified in the ACM) are exchanged: Position values of valves and the temperature value as feedback value.

The process model also includes a secondary circuit, which is coupled via a heat exchange to the primary circuit. By manipulating the valves in the primary circuit, the temperature in the secondary circuit has to be kept in an allowed band.

Although the model has a reduced complexity, compared to the real system, the thermodynamical calculations require high calculation power. The Dymola simulation model is hardly able to run at nominal speed. To allow synchronous exectution of Dymola and the runtime environment FORTE, the time advancement rates of both execution systems are synchronised. Hence, reliable simulation results, which meet the real plant behaviour, have been reached.

With this test case the feasibility of setting up a co-simulation environment, including external simulation tools, is shown. Nevertheless, methods for interaction cannot be generalized, as they are specific to each simulation tool.

5.5 Conclusion and Summary

The selected test cases cover all relevant features of the proposed engineering and simulation workflow. Furthermore, different fields of manufacturing and production system design have been included in the evaluation.

Starting from different domain specific engineering tools data is extracted and stored in the central engineering model. The proposed models have been implemented and are provided by an Eclipse-based engineering environment. As there exist many discipline or domain specific tools it is not feasible to integrate all of them. Hence, the evaluation is based on the integration of selected external tools (i.e. StarUML, ViroCon, Dymola). All required model transformation rules for the information extraction have been prototypically implemented and included in a model transformation workflow. Their functionality has been proven within the selected test cases.

At the first look the engineering effort seems to be slightly higher in comparison to other design methodologies. However, the effort is only shifted towards a collaborative specification phase, which is placed early in the overall engineering cycle. Other acitivities, like documentation, can be shortened due to the explicit, multi-disciplinary specification. Furthermore, the strong focus on well defined components, with stable interfaces fosters an easier reuse of once specified components. Hence, it is likely that the engineering effort is further reduced, if a comprehesive component library is available.

As shown for the robotics test case, also COTS components, like the Schunk PowerCubes, can be easily integrated in the approach. Even more, the internal specification and implementation can be hidden and thus the IP for the encapsulated functionalities can be protected. Only the behaviour at the interfaces needs to be specified, which has to be provided to customers anyway.

From the component-based specification of the production system, including both automation system independent aspects and the automation execution system description, executable code is created. This process can be highly automated. However, in case of faults in the automated plant, the maintenance personnel will usually work directly on the generated applications to keep downtimes short. Hence, there is the need for well-readible automatic control applications. The implemented and presented model-transformations provide hierarchically structured automatic control applications. Furthermore, the clear separation of automatic control functionalities and hardware access facilitates the diagnosis.

Even more, behaviour simulation is made accessible as a helpful engineering support tool with neglectable effort. For industrial projects interfaces and behaviour at the interfaces have to be documented anyhow. However, the late documentation is replaced by an early specification in the proposed workflow. The gathered information is the core of the automatically generated plant simulation application.

The generated plant simulation applications follow the same design guideline as the generated automatic control applications. Hence, the interfaces to the automatic control are clearly separated from the simulation logic.

The test cases show that the plant simulation applications are scaling in the same way as the related automatic control applications. The execution system for automatic control systems is composed of multiple control devices to meet requirements with respect to computational power. The hierarchical structure allows to easily deploy parts of the automatic control application to different devices. This leads to the assumption that also the plant simulation applications scale well.

To connect the automatic control application to the simulated plant (i.e. the plant simulation application) only the hardware access FBs have to be replaced by simulation access FBs. This allows validation and testing of the automatic control functionalities, even if the real, physical plant is not (yet) accessible. In such scenarios both applications (i.e. plant simulation and automatic control) can be deployed to the same devices. Variable time advancement can then speed up the behaviour simulation.

Even external simulation tools can be included. They can either be accessed from replaced hardware access FBs, like shown for the process technology test case, or from the plant simulation application. The later would allow to set up and coordinate more comprehensive co-simulation scenarios.

Switching between simulation and operation is easily possible by just deploying the automatic control application to a different resource. The simulation resource in FORTE makes the necessary switching of hardware access FBs automatically. For the validation of the influence of the communication network, the network simulator ns3 has been included in the simulation execution system. Parts of the automatic control application can be deployed to virtual devices, which are located within the ns3 environment. The easy switching between simulation and operation also facilitates hybrid simulation.

The results of all these simulation activities, which are facilitated by the proposed workflow, lead to a higher efficiency in the commissioning phase, especially for changes of existing plants.

CHAPTER 6

Conclusion and Outlook

6.1 Conclusion

The trend towards mass-customization, high numbers of variants and descreasing product life cycles is evident for consumer goods (e.g. cellular phones). On the other hand, production systems are durable and expensive plants with long-term depreciation. Hence, the production systems will undergo multiple reconfigurations within their life cycles. Each reconfiguration task, like the initial design and development phase, is a multi-disciplinary one. For that reason efficiency gains in the engineering process are crucial. Automatic control design and implementation already make up for more than 50% of the project costs in some domains. Hence, it is important to include automatic control engineering early in the development cycle for beeing able to significantly reduce the effort. Gaps and long established interfaces between the involved disciplines have to be reduced, to foster a fruitful collaboration of experts from the involved disciplines.

This thesis presents a model-based engineering approach for the collaborative design of automated production systems. The model-based and component oriented development paradigms of software engineering are already well accepted for business IT development projects. The presented work is a first, but significant step to introduce these design and development paradigms in the multi-disciplinary environment of automatic control systems design.

A comprehensive engineering workflow is presented in Chapter 3. As a first step elementary building blocks of plants—graspable units named Automation Components (ACs)—are introduced. Interfaces between ACs, but also between the different disciplines within a single AC, have to be specified in a collaborative manner. The definition of the Automation Component Model

(ACM) follows the principle of separation of concerns. Multiple, linked engineering models provide facilities to gather different aspects of the provided engineering data. For a smooth transition from existing workflows a continued use of discipline specific engineering tools and their models for the specification is foreseen. Model transformation rules for selected tools are included in the modelling and configuration environment MECE, which has been an outcome of the MEDEIA project. After the specification of the automated system (including both the plant and control functionality) is completed, the presented workflow promotes the automatic code generation. The implementation of a code generator for the event-based automatic control devices compliant to the IEC 61499 standard is shown. It follows the structuring guidelines for well structured and readible code. Hence, experienced automatic control designers can easily grasp the functionality of the generated automatic control application. This is important in the commissioning phase but also for maintenance.

Simulation of the plant behaviour is presented as a methodology for the validation of automatic control applications in Chapter 4. A simulation framework on the basis of IEC 61499 is presented. The event-based execution semantics of IEC 61499 compliant runtime environments make them highly suited also for discrete event simulation. A huge advantage of using industrial automatic control runtime environments for simulation is the easy coupling of the automatic control application with the plant behaviour simulation. Four scenarios, full simulation, hybrid simulation, simulation with the inclusion of external simulation tools, and normal operation (or testing) are presented. The clear separation of hardware access from automatic control functionality (i.e. hardware access Function Blocks (FBs)) prove to facilitate switching from simulation to hardware access and vice versa. The simulation resource, provided for the open source runtime environment FORTE, even provides an automatic mechanism to switch to simulation coupling without the need to modify the automatic control application. The feasibility of a distributed, component based simulation execution with FORTE is shown. This opens the possibility to distribute the automatic control application and the related plant simulation application onto multiple automatic control devices. Interaction between different simulation components is limited to the modeling of physical interactions. Physical interactions (e.g. kinematic chains, material flow) are well handled by the proposed and presented simulation framework. Two external simulation tools, Dymola and the network simulator ns3, are also included in the simulation execution system.

The integration of the proposed simulation approach into the model-based engineering workflow reveals further advantages. For documentation purposes the interfaces between automatic control applications and the controlled process need to be specified. Timed state charts are chosen to model the plant behaviour within the ACM. Model-transformation facilities are implemented to automatically generate portable IEC 61499 plant simulation applications from the provided specification. These simulation applications can be used to validate the automatic control applications before deploying them to the real plant. Hence, the quality of the automatic control applications can be ensured earlier and the cost-intensive commissioning phase can be shortened.

Experiments on three different test cases from different domains are conducted to validate the proposed design and simulation methodology. First, the modelling and specification workflow is tested. Engineering data is extracted from two different tools reliably. Previously defined components can be re-used easily. Second, executable automatic control applications and plant simulation applications are generated automatically from the models. These can be opened with two different engineering tools, the 4DIAC-IDE and FBDK. Third, different simulation scenarios are executed. The simulation of physical connections works reliably.

After implementing and testing the proposed approach, the research questions can be answered as follows:

RQ1: *Is a model-driven approach feasible for the engineering of manufacturing systems?* Yes: In future manufacturing systems will be changed multiple times during their lifecycle in order to being able to produce the desired products. For that reason a systematic engineering methodology helps to reduce the engineering effort and thus costly down-times. The evaluation of the proposed engineering workflow with the help of test cases from different domains clearly shows the feasibility of such a model-based approach.

RQ2: *Is MDA apt to build the basis for an integrative engineering approach, parallelizing work of experts from multiple disciplines?* Yes: The separation of engineering concerns and the proposed modelling infrastructure with multiple models foster the multi-disciplinary specification and implementation work. The Automation Components provide the interfaces within the disciplines. Furthermore, special attention is put to the interfaces between the involved disciplines. As long as these stay unchanged, the work can be done parallely.

RQ3: *Is it possible to validate by simulation and deploy the same control application without change?* Yes: The provision of two sets of hardware access FBs—one providing process interfaces, the other providing access to the plant simulation—in conjunction with the simulation resource enable this feature. Switching between simulation and operation is done by deploying the application, or parts of the application, to a different type of IEC 61499 resource (i.e. EMB_RES or SIM_EMB_RES).

RQ4: *Is an event-based automatic control runtime environment apt for the execution of distributed, discrete event simulation?* Yes: IEC 61499 (or compliant runtime environments like FORTE) provide all significant features of discrete event simulation environment. Missing features, like time advancement regulation (at fixed or variable rate) can be implemented as extensions. Hence, a runtime environment that is capable to run automatic control applications and simulation applications likewise can be created.

RQ5: *Is it possible to integrate other simulation tools into a co-simulation environment (e.g. network simulator, continuous process simulator) efficiently?* Both ns3 and Dymola were included in test scenarios. For network simulation only different devices—virtual devices within the simulated (i.e. virtual) network—have to be used as target during the deployment. Interaction with real devices is still possible. For the interaction with Dymola only communication FBs and some FBs for en-/decoding of the exchanged values have to be included in the plant interface FBs (or the plant simulation application). Although the integration of these distinct tools is working efficiently, it is not possible to make a general statement about the possibility to integrate any other simulation tool in the proposed co-simulation environment. However, it seems feasible to integrate also other external simulation tools, if a communication channel between the external simulation tool and the co-simulation environment based on FORTE can be established, data-types are compatible, and simulation time can be kept synchron (e.g. all entities running at nominal time or time synchronisation facilities).

The author is confident that the work presented in this thesis is an important step towards a more efficient engineering of production systems. This new approach covers all aspects from collaborative specification to the commissioning of the plants. It can be used for either the initial setup of a new plant or for reconfiguration of existing plants. Different validation scenarios, applicable in various development phases, may be used to increase the quality of the developed automatic control functionality. The simulation framework on basis of the IEC 61499 compliant runtime environment FORTE helps to validate automatic control applications directly on the selected execution platform. Hence, platform aspects, which might have influence on the overall behaviour of the automated system, are included in the validation process.

6.2 Outlook

The presented work is a first step towards an efficient engineering workflow for production systems. The inclusion of simulation in the engineering workflow helps to increase the quality of the automatic control applications and also decrease the overall engineering time frame to deployment. The selected test cases, steming from distinct domains, prove the feasibility of the presented approach in large areas of production systems design. Nevertheless, there are still open points that need attention and further investigation before a full industrial use of the presented concepts is possible.

Modelling Environment

The central engineering models have been well thought of. Nevertheless, additional information (e.g. mechanical data) has to be included and further links within the models have to be established. For better acceptance, also additional specification tools have to be included in the workflow. This requires a thorough analysis of the data provided by the tools. Model transformation rules have to be created for data extraction.

A possible step towards the integration of additional tools could be a two-phased model transformation process for the information extraction. An intermediate model could act as common point for multiple, similar engineering tools. Hence, the effort for creating model transformation rules could be significantly reduced.

Besides the integration of additional third party tools also the creation of a Domain Specific Language (DSL) for the hierarchical modelling, especially using already existing components, is urging for the successful adoption of the proposed engineering approach. Support for the explicit specification of errors and error-handling mechanisms is also missing. Instead of specifying these aspects with the nominal behaviour of an automated component, a separation of concerns should be reached. A separate sub-workflow and the integration of specialized tools should be aimed at. Nonetheless, such specified error-handling mechanisms have to be considered during the code generation and included in the automatic control applications.

The current practice for the maintenance of automation systems, which is focused on getting the plant operational as soon as possible, is to fix faults directly on the controllers in the plant. PLC programming environments are able to detect modifications and the upload functionality can be used to synchronize the deployed programme with the engineering tool. Correctly extracting information from freely programmed code (for PLCs and other control devices) to bring it into structured engineering models is hardly possible. However, at least support functionalties to link the generated code with the models has to be available in a model-based engineering approach.

External Simulation Tools

In some of the involved disciplines simulation models and tools are applied for the analysis of design alternatives. However, only the selected solution is provided to the other disciplines. The presented model-based engineering workflow supports the creation of multiple ACs which provide the same or similar functionality. Possible design alternatives should be included in the central engineering model repository. The alternative solutions are variation points in the design of the automated system. A selection of the AC to being implemented shall be done collaboratively. Nevertheless, additional data needs to be included in these model to satisfy the requirements. The model-based engineering infrastructure should not be a parallel system, but the central system. Hence, external simulation models and results should be integrated in the models as well.

The improvement and extension of the simulation tool independent plant behaviour model, could be further investigated. Including more data in this model would further facilitate a simulation tool independent modelling. Similar to the development of automatic control applications the tool-specific simulation models could be generated via model-transformation.

Another possibility for integrating external simulation tools in the development cycle for production systems could be the establishment of a specialized middleware for production system simulation. Unified interfaces, datatypes, and object definitions (e.g. material) could facilitate the combiniation of arbitrary simulation and control environments.

Simulation Framework

For the validation of larger, and distributed automatic control application with means of distributed simulation on the same control devices the time management functionalities have to be improved. Currently all devices run at the same clock advancement rate. For hybrid simulation scenarios this is not a problem, as the operational parts run at nominal speed anyway. The synchronisation of the Future Event List on the involved control devices could help to speed up the simulation runs for full simulation scenarios. If external simulation tools are included, the synchronisation facility also has to include them for reliable results.

Simulation results are currently provided via log-files and the integration into SCADA systems. Hence, the simulation results have to be evaluated by an expert on the basis of these historic data. To further foster the collaboration of multiple disciplines the visual presentation of results, directly included in the simulation/automatic control runtime environment could help. Game graphics engines provide according APIs which could be used to provide a 3D simulation environment. Furthermore, such engines also include realistic physics models and simulations, which could be used for collission detection and other purposes. Compared to available 3D Simulation applications these game engines provide more open APIs.

Simulation-based Diagnosis

The availablity of plant simulation on the control devices is an enabler for model-based diagnosis in automated systems in the production domain. The need for additional devices, configuration of communication interfaces and the like is compensated. The introduction of model-based diagnosis on a broad basis can help to improve the maintenance of the plants, decrease outages and downtimes and hence operate the production systems more efficiently.

Bibliography

[1] Manufuture High-Level Group, "Strategic Research Agenda: Assuring the future of manufacturing in europe," Manufuture Platform, European Commission, Tech. Rep., Sep. 2006.

[2] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, A. G. Ulsoy, and H. V. Brussel, "Reconfigurable Manufacturing Systems," *Annals of the CIRP*, vol. 48, no. 2, pp. 527–540, 1999.

[3] M. G. Mehrabi, A. G. Ulsoy, and Y. Koren, "Reconfigurable manufacturing systems: Key to future manufacturing," *Journal of Intelligent Manufacturing*, vol. 11, no. 4, pp. 403–419, Aug. 2000.

[4] A. Hirzle, "AutomationML," Press Conference, Hannover Messe - HMI, DaimlerChrysler AG, 2007.

[5] VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik (GMA), Fachbereich Engineering und Betrieb automatisierter Anlagen, "VDI/VDE 3695 Part 3: Engineering of industrial plants, Evaluation and optimization, Subject methods," Dec. 2010.

[6] M. P. Groover, *Automation, Production Systems, and Computer-Integrated Manufacturing*, 3rd ed. Upper Saddle River, NJ: Pearson Education Inc., 2008.

[7] J. Drolet, G. Abdulnour, and M. Rheault, "The cellular manufacturing evolution," *Computers & Industrial Engineering*, vol. 31, no. 1 - 2, pp. 139 – 142, 1996.

[8] J. Browne, D. Dubois, K. Rathmill, S. P. Sethi, and K. E. Stecke, "Classification of flexible manufacturing systems," *The FMS magazine*, vol. 2, no. 2, pp. 114–117, 1984.

[9] H. A. ElMaraghy, “Flexible and reconfigurable manufacturing systems paradigms,” *International Journal of Flexible Manufacturing Systems*, vol. 17, no. 4, pp. 261–276, Oct. 2005.

[10] P. R. Panchalavarapu and V. Chankong, “Design of cellular manufacturing systems with assembly considerations,” *Computers & Industrial Engineering*, vol. 48, no. 3, pp. 449 – 469, 2005.

[11] H. Ford and S. Crowther, *My life and work.* Doubleday, Page & Company, 1922.

[12] M. Fletcher, R. W. Brennan, and D. H. Norrie, “Modeling and reconfiguring intelligent holonic manufacturing systems with Internet-based mobile agents,” *Journal of Intelligent Manufacturing*, vol. 14, no. 1, pp. 7–23, Feb. 2003.

[13] J. H. Christensen, “Holonic Manufacturing Systems: Initial Architecture and Standards Directions,” in *Proceedings of the 1st Euro Workshop on Holonic Manufacturing Systems*, HMS Consortium, Ed., Hannover, Dec. 1994.

[14] R. F. Babiceanu and F. F. Chen, “Development and applications of holonic manufacturing systems: a survey,” *Journal of Intelligent Manufacturing*, vol. 17, no. 1, pp. 111–131, Feb. 2006.

[15] N. R. Jennings and S. Bussmann, “Agent-based control systems: Why are they suited to engineering complex systems?” *Control Systems, IEEE*, vol. 23, no. 3, pp. 61 – 73, Jun. 2003.

[16] M. Merdan, “Knowledge-based Multi-Agent Architecture Applied in the Assembly Domain,” Doctoral Thesis, Vienna University of Technology, Automation and Control Institute, Mar. 2009.

[17] W. Lepuschitz, A. Zoitl, M. Vallée, and M. Merdan, “Toward self-reconfiguration of manufacturing systems using automation agents,” *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 41, no. 1, pp. 52 –69, Jan. 2011.

[18] P. Leitao and F. J. Restivo, “Implementation of a holonic control system in a flexible manufacturing system,” *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 38, no. 5, pp. 699 –709, Sep. 2008.

[19] A. W. Colombo, R. Schoop, and R. Neubert, “An agent-based intelligent control platform for industrial holonic manufacturing systems,” *Industrial Electronics, IEEE Transactions on*, vol. 53, no. 1, pp. 322 – 337, Feb. 2006.

[20] W. Zhang and S. Xie, "Agent technology for collaborative process planning: a review," *The International Journal of Advanced Manufacturing Technology*, vol. 32, pp. 315–325, 2007.

[21] M. Schenk and S. Wirth, *Fabikplanung und Fabrikbetrieb, Methoden für die wandlungsfähige und vernetzte Fabrik.* Springer-Verlag Berlin, 2004.

[22] O. Battaïa and A. Dolgui, "A taxonomy of line balancing problems and their solutionapproaches," *International Journal of Production Economics*, vol. 142, no. 2, pp. 259 – 277, 2013.

[23] C. Becker and A. Scholl, "A survey on problems and methods in generalized assembly line balancing," *European Journal of Operational Research*, vol. 168, no. 3, pp. 694 – 715, 2006.

[24] A. Al-Ahmari and K. Ridgway, "An integrated modelling method to support manufacturing systems analysis and design," *Computers in Industry*, vol. 38, pp. 225–238, 1999.

[25] A. Gunasekaran and Y. Y. Yusuf, "Agile manufacturing: a taxonomy of strategic and technological imperatives," *International Journal of Production Research*, vol. 40, no. 6, pp. 1357–1385, 2002.

[26] A.-W. Scheer, *CIM. Computer Integrated Manufacturing: Der computergesteuerte Industriebetrieb*, 4th ed. Berlin et al.: Springer-Verlag, 1990.

[27] T. Tolio, M. Sacco, W. Terkaj, and M. Urgo, "Virtual factory: An integrated framework for manufacturing systems design and analysis," *Procedia CIRP*, vol. 7, pp. 25 – 30, 2013.

[28] U. Bracht and T. Masurat, "The digital factory between vision and reality," *Computers in Industry*, vol. 56, no. 4, pp. 325 – 333, 2005.

[29] AutomationML initiative, "AutomationML website," (last access: 15.08.2011). [Online]. Available: http://www.automationml.org

[30] R. Drath, Ed., *Datenaustausch in der Anlagenplanung mit AutomationML*, ser. VDI-Buch. Heidelberg, Dordrecht, London, New York: Springer, 2010.

[31] R. Drath, J. Peschke, and S. Lips, "AutomationML Top Level Architecture Document," AutomationML consortium, Tech. Rep., Apr. 2008.

[32] M. Cohn, *Succeeding with Agile - Software Development using Scrum*. Upper Saddle River, NJ: Addison-Wesley, 2011.

[33] K. Beck, *Extreme Programming*. München: Addison-Wesley, 2000.

[34] VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik (GMA), Fachbereich Engineering und Betrieb automatisierter Anlagen, "VDI/VDE 3695 Part 2: Engineering of industrial plants, Evaluation and optimization, Subject processes," Nov. 2010.

[35] M. Abramovici and O. Sieg, "Status and development trends of product lifecycle management systems," in *Proceedings of IPPD'2002*, Nov. 2002.

[36] P. Mohagheghi, R. Conradi, O. Killi, and H. Schwarz, "An empirical study of software reuse vs. defect-density and stability," in *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*, 2004, pp. 282–291.

[37] B. Meyer, "The grand challenge of trusted components," in *Software Engineering, 2003. Proceedings. 25th International Conference on*, 2003, pp. 660–667.

[38] G. Pietrek and J. Trompeter, Eds., *Modellgetriebene Softwareentwicklung*. Frankfurt: entwickler.press, 2007.

[39] T. Stahl, M. Völtner, S. Efftinge, and A. Haase, *Modellgetriebene Softwareentwicklung*, 2nd ed. Heidelberg: dpunkt.verlag, 2007.

[40] S. Sendall and W. Kozaczynski, "Model transformation: the heart and soul of model-driven software development," *Software, IEEE*, vol. 20, no. 5, pp. 42–45, 2003.

[41] J. Sztipanovits and G. Karsai, "Model-integrated computing environment," *ACM SIGSOFT Software Engineering Notes*, vol. 22, pp. 72–73, Sep. 1997.

[42] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty, "Model-integrated development of embedded software," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 145 – 164, Jan. 2003.

[43] Vanderbuilt University, "Institute for Software Integrated Systems," (last access: 03.12.2011). [Online]. Available: http://www.isis.vanderbilt.edu

[44] K. Czarnecki and U. W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*. Boston, MA, USA: Addison-Wesley, 2000.

[45] D. Abrahams and A. Gurtovoy, *C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond*, ser. The C++ In-Depth Series. Boston, MA, USA: Addison-Wesley Professional, 2004.

[46] Object Management Group, "Object management group (omg) website," (last access: 03.12.2012). [Online]. Available: http://www.omg.org/

[47] Object Management Group, "Model-Driven Architecture (MDA) website," (last access: 03.12.2012). [Online]. Available: http://www.omg.org/mda

[48] OMG Architecture Board, "Technical Guide to Model Driven Architecture: The MDA Guide v1.0.1," Object Management Group (OMG), Tech. Rep., Jun. 2003.

[49] OMG Platform Technology Committee, OMG Domain Technology Committee, and OMG Architecture Board, "Model Driven Architecture (MDA)," Object Management Group, Tech. Rep., Jul. 2001.

[50] M. Fowler, *Domain-Specific Languages*. Upper Saddle River, NJ: Addison-Wesley, 2011.

[51] C. Szyperski, *Component Software Beyond Object-Oriented Programming*, 2nd ed. Addison-Wesley, 2002.

[52] J. Yang and M. P. Papazoglou, "Interoperation support for electronic business," *Commun. ACM*, vol. 43, no. 6, pp. 39–47, Jun. 2000.

[53] H. Sneed, "Integrating legacy software into a service oriented architecture," in *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on*, 2006, pp. 11 pp.–14.

[54] L. de Alfaro and T. A. Henzinger, "Interface theories for component-based design," in *Proceedings of the First International Workshop on Embedded Software, EMSOFT 2001*, ser. LNCS 2211, Oct. 2001.

[55] IEC TC65/WG6, *Programmable controllers – Part 3: Programming languages*. Geneva: International Electrotechnical Commission (IEC), 1993.

[56] R. Lewis, *Programming industrial control systems using IEC 1131-3*, 2nd ed., ser. IEE control engineering series. London, UK: Institution of Electrical Engineers, 1998.

[57] K.-H. John and M. Tiegelkamp, *IEC 61131-3: Programming Industrial Automation Systems*, 2nd ed. Berlin, Heidelberg: Springer Verlag, 2010.

[58] C. Johnsson and K.-E. Årzén, "Grafchart and grafcet: a comparison between two graphical languages aimed for sequential control applications," in *Preprints 14th World Congress of IFAC, vol. A*, 1999, pp. 19–24.

[59] R. Lewis, *Modelling control systems using IEC 61499 – Applying function blocks to distributed systems*. London, UK: The Institution of Electrical Engineers, 2001.

[60] IEC TC65/WG6, *IEC 61499: Function blocks for industrial-process measurement and control systems – Part 1: Architecture*. Geneva: International Electrotechnical Commission (IEC), 2005.

[61] L. H. Yoong, P. Roop, V. Vyatkin, and Z. Salcic, "A synchronous approach for iec 61499 function block implementation," *Computers, IEEE Transactions on*, vol. 58, no. 12, pp. 1599–1614, 2009.

[62] A. Zoitl, "Basic Real-Time Reconfiguration Services for Zero Down-Time Automation Systems," Doctoral Thesis, Vienna University of Technology, Automation and Control Institute, Nov. 2007.

[63] C. Sünder, "Evaluation of downtimeless system evolution in automation and control systems," Doctoral Thesis, Vienna University of Technology, Automation and Control Institute, 2008.

[64] Y. Alsafi and V. Vyatkin, "Ontology-based reconfiguration agent for intelligent mechatronic systems in flexible manufacturing," *Robotics and Computer-Integrated Manufacturing*, vol. 26, no. 4, pp. 381–391, 2010.

[65] M. Vallee, M. Merdan, W. Lepuschitz, and G. Koppensteiner, "Decentralized reconfiguration of a flexible transportation system," *Industrial Informatics, IEEE Transactions on*, vol. 7, no. 3, pp. 505 –516, Aug. 2011.

[66] J. Banks, J. S. Carson II, B. L. Nelson, and D. M. Nicol, *Discrete-Event System Simulation*, 5th ed. Upper Saddle River, NJ: Prentice Hall, 2010.

[67] P. Bratley, B. L. Fox, and L. E. Schrage, *A Guide to Simulation*. New York, Berlin, Heidelberg, Tokyo: Springer-Verlag, 1983.

[68] W. Kelton, R. Sadowski, and N. Swets, *Simulation with Arena*, 5th ed. McGraw-Hill Science, 2009.

[69] VDI-Gesellschaft Fördertechnik Materialfluss Logistik, "VDI 3633 Part 8: Simulation of systems in material handling, logistics and production - Machine-oriented simulation," Apr. 2007.

[70] P. Klingstam and P. Gullander, "Overview of simulation tools for computer-aided production engineering," *Computers in Industry*, vol. 38, no. 2, pp. 173–186, Mar. 1999.

[71] Rockwell Automation, "Arena website," (last access: 03.12.2012). [Online]. Available: http://www.arenasimulation.com

[72] Siemens PLM, "Siemens plant simulation," (last access: 03.12.2012). [Online]. Available: http://www.plm.automation.siemens.com/en_us/products/tecnomatix/plant_design/plant_simulation.shtml

[73] INCONTROL Simulation Solutions, "Enterprise Dynamics," (last access: 12.07.2012). [Online]. Available: http://www.incontrolsim.com

[74] XJ Technologies Company, "Anylogic," (last access: 12.07.2012). [Online]. Available: http://www.xjtek.com/

[75] F. Himmler and M. Amberg, "Die Digitale Fabrik - eine Literaturanalyse," in *Wirtschaftsinformatik Proceedings 2013*, 2013.

[76] R. J. Schack, "Methodik zur bewertungsorientierten Skalierung der Digitalen Fabrik," Ph.D. dissertation, Technische Universität München, 2007.

[77] European Factories of the Future Research Association (EFFRA), "EFFRA web-site," (last access: 4.1.2014). [Online]. Available: http://www.effra.eu/

[78] B. Svensson, "Optimisation of Manufacturing Systems Using Time Synchronised Simulation," Department of Signals and Systems Automation Research Group, Chalmers University of Technology, Göteborg, Sweden, Licentiate Thesis, May 2010.

[79] L. G. Barajas, S. R. Biller, F. Gu, and C. Yuan, "Virtual launch & validation of manufacturing automation controls," in *Automation Science and Engineering (CASE), 2010 IEEE Conference on*, Aug. 2010, pp. 412 –419.

[80] F. Auinger, M. Vorderwinkler, and G. Buchtela, "Interface driven domain-independent modeling architecture for "soft-commissioning" and "reality in the loop"," in *Winter Simulation Conference 1999*, 1999, pp. 798–805.

[81] M. Barth, A. Fay, F. Wagner, and G. Frey, "Effizienter Einsatz Simulations-basierter Tests in der Entwicklung automatisierungstechnischer Systeme," in *Kongress Automation 2010, VDI-Berichte 2092*, Jun. 2010, extended version available at http://www.aut.uni-saarland.de/uploads/media/MD_AF_KonAutom_jun_2010.pdf.

[82] dSPACE GmbH, "dSPACE website," (last access: 18.08.2011). [Online]. Available: http://www.dspace.com

[83] National Instruments, "National instruments website," (last access: 03.12.2012). [Online]. Available: http://www.ni.com

[84] F. Gu, W. S. Harrison, D. M. Tilbury, and C. Yuan, "Hardware-In-The-Loop for Manufacturing Automation Control: Current Status and Identified Needs," in *3rd Annual IEEE Conference on Automation Science and Engineering*, Sep. 2007, pp. 1105–1110.

[85] D. Maclay, "Simulation gets into the loop," *IEE Review*, vol. 43, no. 3, pp. 109 –112, May 1997.

[86] O. Lenord and M. Brohm, "Maschinensimulation - Eckstein des Simultaneous Engineering im Bereich Steuerungstechnik," in *SPS/IPC/Drives Kongress 2007*, 2007, pp. 537–546.

[87] D. Scheifele, U. Eger, S. Röck, and P. Sekler, "Potentiale der Hardware-in-the-Loop Simulation für Maschinen und Anlagen," in *SPS/IPC/Drives Kongress 2007*, 2007, pp. 555–565.

[88] H. Schludermann, T. Kirchmair, and M. Vorderwinkler, "Soft-commissioning: hardware-in-the-loop-based verification of controller software," in *Proceedings of the 32nd conference on Winter simulation (WSC 2000)*, 2000, pp. 893–899.

[89] F. Maturana, R. J. Staron, D. Carnahan, A. Iype, and K. Hall, "Holonic multi-agent system for real time simulation of control systems," in *5th International Conference on Industrial Applications of Holonic and Multi-Agent Systems, HoloMAS 2011*, ser. LNAI, V. Marik, P. Vrba, and P. Leitao, Eds., vol. 6867. Springer Heidelberg, Aug. 2011.

[90] L. Ferrarini and A. Dedè, "A model-based approach for mixed hardware in the loop simulation of manufacturing systems," in *10th IFAC Workshop on Intelligent Manufacturing Systems (IMS 10)*, 2010.

[91] M. Ogurek, J. Alex, and M. Schütze, "Simulation als Basis für Entwicklung, Test und fehlerarme Inbetriebnahme von Automatisierungssystemen komplexer Prozesse," in *10. Fachtagung Entwurf komplexer Automatisierungssysteme (EKA 2008)*, Apr. 2008, pp. 331 – 341.

[92] The MathWorks, Inc, "MATLAB/Simulink website," (last access: 27.11.2013). [Online]. Available: http://www.mathworks.com/products/simulink/

[93] G. Frey and L. Liu, "Modellierung und Simulation vernetzter Automatisierungs- und Regelungssysteme in Modelica," *at - Automatisierungstechnik*, vol. 57, no. 9, pp. 466 –476, 2009.

[94] L. Liu and G. Frey, "Feasibility analysis for networked control systems by simulation in modelica," in *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, Sep. 2008, pp. 729 –732.

[95] ABB Ltd, "ABB Robot Studio," (last access: 06.07.2012). [Online]. Available: http://www.abb.com/product/seitp327/78fb236cae7e605dc1256f1e002a892c.aspx

[96] KUKA Roboter GmbH, "KUKA.sim," (last access: 12.07.2012). [Online]. Available: http://www.kuka-robotics.com/germany/en/products/software/simulation/start.htm

[97] S. Kain, M. Merz, M. Reichl, and C. Heuschmann, "Anlagensimulation in IEC 61131," in *Tagungsband ASIM STS/GMMS Workshop 2009*, Mar. 2009.

[98] S. Jain, C. Yuan, and P. Ferreira, "EMBench: A Rapid Prototyping Environment for Numerical Control Systems," in *ASME 2002 International Mechanical Engineering Congress and Exposition (IMECE2002)*, Nov. 2002.

[99] C. Yuan and P. Ferreira, "An integrated rapid prototyping environment for reconfigurable manufacturing systems," *Proceedings of 2003 ASME International Mechanical Engineering Congress and Exposition (IMECE 2003), Washington, DC, USA*, 2003.

[100] M. N. Rooker, T. Strasser, G. Ebenhofer, M. Hofmann, and R. V. Osuna, "Modeling flexible mechatronical based assembly systems through simulation support," in *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, Sep. 2008, pp. 452 – 455.

[101] W. S. Harrison, D. M. Tilbury, and C. Yuan, "From hardware-in-the-loop to hybrid process simulation: An ontology for the implementation phase of a manufacturing system," *Automation Science and Engineering, IEEE Transactions on*, vol. 9, no. 1, pp. 96 –109, Jan. 2012.

[102] Pabadis'Promise Consortium, "Work package 8: System simulation – deliverable 8.3: Common demonstration system," PABADIS'PROMISE project, Tech. Rep., Dec. 2008.

[103] E. Estévez, M. Marcos, and D. Orive, "Automatic generation of PLC automation projects from component-based models," *The International Journal of Advanced Manufacturing Technology*, vol. 35, no. 5-6, pp. 527–540, 2007.

[104] S. Dominka and F. Schiller, "Hybride Inbetriebnahme von Produktionsanlagen - von virtuell zu real," in *SPS/IPC/Drives Kongress 2007*, 2007, pp. 527–535.

[105] A. Sulistio, C. S. Yeo, and R. Buyya, "A taxonomy of computer-based simulations and its mappings to parallel and distributed simulation tools," *Softw. Pract. Exper.*, vol. 34, no. 7, pp. 653–673, Jun. 2004.

[106] T. M. Baker, "Time management in distributed simulation models," in *Proceedings of Fourth International Simulation Technology and Training Conference (SimTecT'99)*, Mar. 1999.

[107] J. Misra, "Distributed Discrete-Event Simulation," *Computing Surveys*, vol. 18, no. 1, pp. 39–65, Mar. 1986.

[108] R. M. Fujimoto, "Parallel and Distributed Simulation Systems," in *Winter Simulation Conference 2001*, 2001, pp. 147–157.

[109] V. Balakrishnan, R. Radhakrishnan, D. M. Rao, N. Abu-Ghazaleh, and P. A. Wilsey, "A performance and scalability analysis framework for parallel discrete event simulators," *Simulation Practice and Theory*, vol. 8, no. 8, pp. 529 – 553, 2001.

[110] P. Lendermann, "About the need for distributed simulation technology for the resolution of real-world manufacturing and logistics problems," in *Proceedings of the Winter Simulation Conference, 2006 (WSC 06)*, Dec. 2006, pp. 1119 –1128.

[111] A. Ferscha and S. K. Tripathi, "Parallel and Distributed Simulation of Discrete Event Systems," University of Maryland, Computer Science Department, Tech. Rep. CS-TR-3336, 1998. [Online]. Available: http://hdl.handle.net/1903/659

[112] A. Fabbri and L. Donatiello, "Coordination languages for parallel discrete event simulation," in *Proceedings of the Thirtieth Hawaii International Conference on System Sciences, 1997*, Jan. 1997, pp. 330 –339.

[113] Y. Tang, K. S. Perumalla, R. M. Fujimoto, H. Karimabadi, J. Driscoll, and Y. Omelchenko, "Optimistic Parallel Discrete Event Simulations of Physical Systems using Reverse Computation," in *Workshop on Principles of Advanced and Distributed Simulation (PADS'05)*, 2005, pp. 26–35.

[114] M. Verhoef, P. Visser, J. Hooman, and J. Broenink, "Co-simulation of distributed embedded real-time control systems," in *Proceedings of the 6th International Conference on Integrated Formal Methods (IFM'07)*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 639–658.

[115] A. Al-Hammouri, M. Branicky, and V. Liberatore, "Co-simulation tools for networked control systems," in *Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, M. Egerstedt and B. Mishra, Eds. Springer Berlin / Heidelberg, 2008, vol. 4981, pp. 16–29.

[116] A. Siemers and D. Fritzson, "A Meta-Modeling Environment for Mechanical System Co-Simulations," in *48th Conference on Simulation and Modelling (SIMS'07)*, 2007.

[117] M. Geimer, T. Krüger, and P. Linsel, "Co-Simulation, gekoppelte Simulation oder Simulatorkopplung? Ein Versuch der Begriffsvereinheitlichung," *O+P Zeitschrift für Fluidtechnik*, vol. 50, no. 11-12, pp. 572–576, Nov. 2006.

[118] A. Amory, F. Moraes, L. Oliveira, N. Calazans, and F. Hessel, "A heterogeneous and distributed co-simulation environment," in *Proceedings of 15th Symposium on Integrated Circuits and Systems Design, 2002*, 2002, pp. 115 – 120.

[119] M. Marcos, U. Gangoiti, E. Estevez, J. Portillo, and I. Calvo, "A CORBA-based co-simulation framework for integrating COTS tools," in *Proceedings of the 6th Portuguese conference on Automatic Control*, Jun. 2004.

[120] S. J. E. Taylor, N. Mustafee, S. Kite, C. Wood, S. J. Turner, and S. Strassburger, "Improving simulation through advanced computing techniques: Grid computing and simulation interoperability," in *Winter Simulation Conference (WSC), Proceedings of the 2010*, Dec. 2010, pp. 216 –230.

[121] IEEE, "IEEE Std. 1516-2010: IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Framework and Rules," Aug. 2010.

[122] S. Strassburger, "The Road to COTS-Interoperability: From Generic HLA-Interfaces Towards Plug-and-Play Capabilities," in *Simulation Conference, 2006. WSC 06. Proceedings of the Winter*, Dec. 2006, pp. 1111 –1118.

[123] A. H. Ng, J. Adolfsson, M. Sundberg, and L. J. De Vin, "Virtual manufacturing for press line monitoring and diagnostics," *International Journal of Machine Tools & Manufacture*, vol. 48, pp. 565 – 575, 2008.

[124] R. Duray, P. T. Ward, G. W. Milligan, and W. L. Berry, "Approaches to mass customization: configurations and empirical validation," *Journal of Operations Management*, vol. 18, no. 6, pp. 605 – 625, 2000.

[125] G. D. Silveira, D. Borenstein, and F. S. Fogliatto, “Mass customization: Literature review and research directions,” *International Journal of Production Economics*, vol. 72, no. 1, pp. 1 – 13, 2001.

[126] A. Hirzle, *Datenaustausch in der Anlagenplanung mit AutomationML*, ser. VDI-Buch. Heidelberg, Dordrecht, London, New York: Springer, 2010, ch. Vorwort von Anton Hirzle.

[127] MEDEIA consortium, “Medeia project website,” (last access: 03.12.2012). [Online]. Available: http://www.medeia.eu

[128] Siemens PLM, *JT Open Technology Factsheet*, 2011.

[129] Object Management Group, “OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1,” Aug. 2011.

[130] MEDEIA consortium, “Deliverable d2.2: Industrial requirements report,” MEDEIA consortium, Tech. Rep., 2008.

[131] Department of Information Technology at Uppsala University (UPP), Sweden and Department of Computer Science at Aalborg University (AAL), Denmark, “Uppaal website,” (last access: 18.10.2012). [Online]. Available: http://www.uppaal.org/

[132] D. Harel, “Statecharts: A visual formalism for complex systems,” *Science of Computer Programming*, vol. 8, no. 3, pp. 231–274, 1987.

[133] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, *Model Checking*. The MIT Press, 1999.

[134] A. Zoitl and V. Vyatkin, “IEC 61499 Architecture for Distributed Automation: The "Glass Half Full" View,” *IEEE Ind. Electron. Mag.*, vol. 3, no. 4, pp. 7 – 23, 2009.

[135] INTERBUS Club, “FDCML 2.0 Specification, Version 1.0,” Nov. 2002.

[136] C. Sünder, A. Zoitl, J. H. Christensen, V. Vyatkin, R. W. Brennan, A. Valentini, L. Ferrarini, T. Strasser, J. L. Martinez Lastra, and F. Auinger, “Usability and interoperability of iec 61499 based distributed automation systems,” in *Industrial Informatics, 2006 IEEE International Conference on*, aug. 2006, pp. 31 –37.

[137] J. Lastra, A. Lobov, and L. Godinho, “Closed loop control using an IEC 61499 application generator for scan-based controllers,” in *Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on*, vol. 1, 2005, pp. 323–330.

[138] M. Didonet Del Fabro and P. Valduriez, "Towards the efficient development of model transformations using model weaving and matching transformations," *Software & Systems Modeling*, vol. 8, no. 3, pp. 305–324, 2009.

[139] M. Barth, R. Drath, A. Fay, F. Zimmer, and K. Eckert, "Evaluation of the openness of automation tools for interoperability in engineering tool chains," in *Proc. of the 17th IEEE International Conference on Emerging Technologies and Factory Automation 2012 (ETFA'12)*, Sep. 2012.

[140] PLCopen, "PLCopen web-site," (last access: 4.1.2014). [Online]. Available: http://www.plcopen.org

[141] R. Drath and M. Barth, "Concept for managing multiple semantics with AutomationML - maturity level concept of semantic standardization," in *Proc. of the 17th IEEE International Conference on Emerging Technologies and Factory Automation 2012 (ETFA'12)*, 2012.

[142] S. M. Falconer, N. F. Noy, and M.-A. Storey, "Towards Understanding the Needs of Cognitive Support for Ontology Mapping," in *International Workshop on Ontology Matching (OM-2006)*, Nov. 2006, pp. 25–36.

[143] S. M. Falconer, N. F. Noy, and M.-A. Storey, "Ontology mapping - a user survey," in *The Second International Workshop on Ontology Matching (OM-2007)*, Nov. 2007, pp. 49–60.

[144] N. F. Noy and M. A. Musen, "The PROMPT suite: interactive tools for ontology merging and mapping," *Int. J. Human-Computer Studies*, vol. 59, pp. 983–1024, 2003.

[145] N. F. Noy, "Semantic Integration: a Survey of Ontology-Based Approaches," *SIGMOD Record*, vol. 33, no. 4, pp. 65–70, Dec. 2004.

[146] L. van Elst and M. Kiesel, "Generating and Integrating Evidence for Ontology Mappings," in *14th International Conference, EKAW 2004*, ser. LNCS, vol. 3257. Springer Berlin / Heidelberg, Oct. 2004, pp. 15–29.

[147] S. M. Falconer and N. F. Noy, "Interactive techniques to support ontology matching," in *Schema Matching and Mapping*, ser. Data-Centric Systems and Applications, Z. Bellahsene, A. Bonifati, and E. Rahm, Eds. Heidelberg, Dordrecht, London, New York: Springer, 2011, ch. 2, pp. 29–51.

[148] E. Rahm, "Towards large-scale schema and ontology matching," in *Schema Matching and Mapping*, ser. Data-Centric Systems and Applications, Z. Bellahsene, A. Bonifati, and E. Rahm, Eds. Heidelberg, Dordrecht, London, New York: Springer, 2011, ch. 1, pp. 3–27.

[149] M. Colla, personal communication, Lugano, Switzerland, Oct. 2009.

[150] D. Vanoverberghe, N. Bjørner, J. Halleux, W. Schulte, and N. Tillmann, "Using dynamic symbolic execution to improve deductive verification," in *Model Checking Software*, ser. Lecture Notes in Computer Science, K. Havelund, R. Majumdar, and J. Palsberg, Eds. Springer Berlin Heidelberg, 2008, vol. 5156, pp. 9–25.

[151] R. Pelánek, "Fighting state space explosion: Review and evaluation," in *Formal Methods for Industrial Critical Systems*, ser. Lecture Notes in Computer Science, D. Cofer and A. Fantechi, Eds. Springer Berlin Heidelberg, 2009, vol. 5596, pp. 37–52.

[152] Timed and hybrid systems group at VERIMAG, "Kronos website," (last access: 18.10.2012). [Online]. Available: http://www-verimag.imag.fr/DIST-TOOLS/TEMPO/kronos/

[153] E. Cagno, F. Caron, and M. Mancini, "Risk analysis in plant commissioning: the multilevel hazop," *Reliability Engineering & System Safety*, vol. 77, no. 3, pp. 309 – 323, 2002.

[154] StarUML Project, "Staruml website," 2013, (last access: 23.04.2013). [Online]. Available: http://staruml.sourceforge.net/

[155] L. Ferrarini, M. Allevi, and A. Dedè, "Design and implementation of an automatic on-line diagnosis with tidiam and tide," in *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, Sep. 2010, pp. 1 –6.

[156] A. Schimmel and A. Zoitl, "Real-time communication for iec 61499 in switched ethernet networks," in *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2010 International Congress on*, oct. 2010, pp. 406 –411.

[157] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Boston: Kluwer Academic Publisher, 1997.

[158] C. Sünder, A. Zoitl, J. H. Christensen, H. Steininger, and J. Fritsche, "Considering iec 61131-3 and iec 61499 in the context of component frameworks," in *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*, july 2008, pp. 277 –282.

[159] J. H. Christensen, "Proxy design pattern for iec 61499," 2010, (last access: 05.12.2012). [Online]. Available: http://www.holobloc.com/doc/despats/proxy/

[160] C. Gerber and H.-M. Hanisch, "Does portability of IEC 61499 mean that once programmed control software runs everywhere?" in *Proc. of 10th IFAC Workshop on Intelligent Manufacturing Systems*, 2010, pp. 29 – 34.

[161] R. Hametner, B. Kormann, B. Vogel-Heuser, D. Winkler, and A. Zoitl, "Test case generation approach for industrial automation systems," in *Automation, Robotics and Applications (ICARA), 2011 5th Int. Conf. on*, Dec. 2011, pp. 57 –62.

[162] F. Wagner, J. Bohl, and G. Frey, "An IEC 61499 interpretation and implementation focused on usability," in *Emerging Technologies and Factory Automation (ETFA), 2008. IEEEInt. Conf. on*, Sept. 2008, pp. 184 –191.

[163] A. Zoitl and H. Prähofer, "Building hierarchical automation solutions in the IEC 61499 modeling language," in *Ind. Informatics (INDIN), 2011 9th IEEE Int. Conf. on*, July 2011, pp. 557 –564.

[164] I. Hegny, T. Strasser, M. Melik-Merkumians, M. Wenger, and A. Zoitl, "Towards an Increased Reusability of Distributed Control Applications Modeled in IEC 61499," in *Emerging Technologies and Factory Automation (ETFA), 2012 IEEE Conference on*, 2012.

[165] G. Čengić and K. Åkesson, "On formal analysis of iec 61499 applications, part b: Execution semantics," *Industrial Informatics, IEEE Transactions on*, vol. 6, no. 2, pp. 145 –154, May 2010.

[166] L. Ferrarini and C. Veber, "Implemenation approaches for the execution model of IEC 61499 applications." in *Proceedings of the 2ndIEEE International Conference on Industrial Informatics, INDIN'04*, Berlin, Jun. 2004, pp. 612–617.

[167] HOLOBLOC, Inc, "FBDK – The Function Block Development Kit," http://www.holobloc.com, May 2010. [Online]. Available: http://www.holobloc.com

[168] ICS Triplex, "ISaGRAF Webpage," http://www.isagraf.com, (last access: 25.12.2013).

[169] A. Zoitl, *Real-Time Execution for IEC 61499*. Durham, North Carolina, USA: International Society of Automation and O^{3}neida, 2009.

[170] A. Zoitl, R. Smodic, C. Sünder, and G. Grabmair, "Enhanced real-time execution of modular control software based on IEC 61499," in *Proceedings of the 2006 IEEE International Conference on Robotics and Automation, ICRA'06*, Orlando, May 2006, pp. 327–332.

[171] 4DIAC initiative, "FORTE (4DIAC-RTE) project web-site," (last access: 3.1.2014). [Online]. Available: http://www.fordiac.org/8.0.html

[172] ABB Ltd, "Abb website," (last access: 20.08.2011). [Online]. Available: http://www.abb.com

[173] KUKA Roboter GmbH, "Kuka-robotics website," (last access: 20.08.2011). [Online]. Available: http://www.kuka-robotics.com

[174] M. Wenger, M. Melik-Merkumians, I. Hegny, R. Hametner, and A. Zoitl, "Utilizing IEC 61499 in an MDA control application development approach," in *Automation Science and Engineering (CASE), 2011 IEEE Conf. on*, Aug. 2011, pp. 495 –500.

[175] V. Vyatkin, "The IEC 61499 standard and its semantics," *IEEE Ind. Electron. Mag.*, vol. 3, no. 4, pp. 40–48, Dec. 2009.

[176] G. Ebenhofer, M. N. Rooker, and S. Falsig, "Generic and reconfigurable IEC 61499 function blocks for advanced platform independent engineering," in *Proc. of 9th IEEE Int. Conf. Ind. Informatics (INDIN)*, 2011, pp. 591–596.

[177] I. Hegny, A. Zoitl, and W. Lepuschitz, "Integration of simulation in the development process of distributed IEC 61499 control applications," in *Industrial Technology (ICIT), 2009 IEEE Int. Conf.on*, Feb. 2009, pp. 1–6.

[178] K. Thramboulidis, "Development of distributed industrial control applications: The corfu framework," in *4th IEEE International Workshop on Factory Communication Systems*, 2002, pp. 39–46.

[179] C. Landsteiner, F. Andren, and T. Strasser, "Evaluation and test environment for automation concepts in Smart Grids applications," in *Proc. IEEE First Int. Smart Grid Modeling and Simulation (SGMS) Workshop*, 2011, pp. 67–72.

[180] NXT Control, GmbH, "NXT Control website," (last access: 07.01.2013). [Online]. Available: http://www.nxtcontrol.com/

[181] I. Hegny and A. Zoitl, "Component-based Simulation Framework for Production Systems," in *International Conference on Industrial Technology*, Mar. 2010.

[182] B. Siciliano, L. Sciavicco, L. Villani, and O. Giuseppe, *Robotics: Modelling, Planning and Control*. London, UK: Springer, 2009.

[183] R. Froschauer, F. Auinger, A. Schimmel, and A. Zoitl, "Engineering of communication links with aadl in iec 61499 automation and control systems," in *Industrial Informatics, 2009. INDIN 2009. 7th IEEE International Conference on*, 2009, pp. 582–587.

[184] Riverbed, "OPNET Modeler Suite," (last access: 30.12.2013). [Online]. Available: http://www.riverbed.com/products-solutions/products/network-performance-management/network-planning-simulation/Network-Simulation.html

[185] University of Southern California, Information Science Institute, "Network Simulator ns-2 website," (last access: 30.12.2013). [Online]. Available: http://nsnam.isi.edu/nsnam/index.php/User_Information

[186] NS3-consortium, "ns-3 web-site," (last access: 30.12.2013). [Online]. Available: http://www.nsnam.org

[187] M. Wenger, R. Hametner, and A. Zoitl, "IEC 61131-3 control applications vs. control applications transformed in IEC 61499," in *10th IFAC Workshop on Intelligent Manufacturing Systems*, Jul. 2010.

[188] SCHUNK GmbH & Co. KG, "Schunk company website," 2013, (last access: 23.04.2013). [Online]. Available: http://www.de.schunk.com/

[189] D. Pilone and N. Pitman, *UML 2.0 in a Nutshell*. O'Reilly, 2005.

[190] B. Oesterreich, *Die UML 2.0 Kurzreferenz für die Praxis*. Oldenbourg Wissenschaftsverlag, 2005.

[191] M. N. Rooker, G. Ebenhofer, M. Wenger, I. Hegny, A. Dedè, M. Colla, and M. Semo, "Deliverable D5.3: MEDEIA Design & Engineering Framework Prototype Implemented & Tested," MEDEIA consortium, Tech. Rep., 2010.

[192] The Eclipse Foundation, "Eclipse web-site," (last access: 2.1.2014). [Online]. Available: http://eclipse.org

[193] The Eclipse Foundation, "Eclipse Modelling Framework (EMF) - project web-site," (last access: 2.1.2014). [Online]. Available: http://projects.eclipse.org/projects/modeling.emf

[194] MEDEIA consortium, "MEDEIA Design and Engineering Framework (MDEF)," Sep. 2010, (last access: 19.03.2013). [Online]. Available: http://sourceforge.net/projects/medeia/

[195] G. Ebenhofer, "2nd Version of the MEDEIA Open Source Modelling Prototype," MEDEIA Project, Public Deliverable D5.2, Sep. 2010.

[196] The Eclipse Foundation, "Modeling Workflow Engine - project website," (last access: 3.1.2014). [Online]. Available: https://projects.eclipse.org/projects/modeling.emf.mwe

[197] 4DIAC initiative, "4DIAC-IDE project web-site," (last access: 3.1.2014). [Online]. Available: http://www.fordiac.org/9.0.html

[198] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software*. Addison-Wesely Verlag, 1996.

[199] French Institute for Research in Computer Science and Automation, "Institute web-site," (last access: 4.1.2014). [Online]. Available: https://www.inria.fr/en/

[200] M. Lacage, F. Urbani, T. Turletti, and W. Dabbous, "Direct Code Execution for ns-3," in *Workshop on ns-3 (WNS3)*, Mar. 2012. [Online]. Available: http://www.nsnam.org/wp-content/uploads/2011/10/ns3DCEPoster1200.pdf

[201] NS3-consortium, "ns-3: Direct Code Execution project web-site," (last access: 4.1.2014). [Online]. Available: http://www.nsnam.org/overview/projects/direct-code-execution/

[202] Dassault Systemes, "Dymola product web-site," (last access: 4.1.2014). [Online]. Available: http://www.3ds.com/products-services/catia/capabilities/systems-engineering/modelica-systems-simulation/dymola

[203] J. H. Christensen, "IEC 61499 Compliance Profile for Feasibility Demonstrations," Holonic Manufacturing Systems Consortium, Tech. Rep., 2010. [Online]. Available: http://holobloc.com/doc/ita/index.htm

[204] F. Wagner, "JPAARealTime," 2007. [Online]. Available: http://www.aut.uni-saarland.de/software/

List of Publications

[I] I. Hegny, "Verteiltes Management für den Betrieb autonomer Energieversorgungssysteme," Master's thesis, Institut für Automatisierungs- und Regelungstechnik, 2006.

[II] I. Hegny, R. Holzer, G. Grabmair, A. Zoitl, F. Auinger, and E. Wahlmüller, "A distributed energy management approach for autonomous power supply systems," in *INDIN 2007 Conference Proceedings*, 2, 2007, pp. 1111–1116.

[III] M. Merdan, A. Zoitl, I. Hegny, and B. Favre-Bulle, "Einsatzanalyse von RFID-technologie in KMU," *ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb*, vol. 102, no. 9, pp. 587–593, 2007.

[IV] M. Merdan, G. Koppensteiner, I. Hegny, and B. Favre-Bulle, "Application of an ontology in a transport domain," in *Proceedings IEEE International Conference on Industrial Technology (ICIT2008)*, 2008.

[V] L. Ferrarini, C. Veber, G. Ebenhofer, M. Rooker, T. Strasser, M. Colla, I. Hegny, C. K. Sünder, and M. Wenger, "State-of-the-art and technology review report," E376 - Institute of Automation and Control; Vienna University of Technology, Tech. Rep., 2008.

[VI] G. Koppensteiner, M. Merdan, I. Hegny, and G. Weidenhausen, "A change-direction-algorithm for distrubuted multi-agent transport systems," in *Proceedings of 2008 IEEE International Conference on Mechatronics and Automation*, 2008.

[VII] M. Merdan, G. Koppensteiner, A. Zoitl, and I. Hegny, "Intelligent-agent based approach for assembly automation," in *Proceedings 2008 IEEE Conference on Soft Computing in Industrial Applications SMCia/08*. IEEE Conference Proceedings, 2008.

[VIII] I. Hegny, O. Hummer-Koppendorfer, A. Zoitl, G. Koppensteiner, and M. Merdan, "Integrating software agents and IEC 61499 real-time control for reconfigurable distributed manufacturing systems," in *Proceedings IEEE Third Symposium on Industrial Embedded Systems SIES'2008*, 2008.

[IX] T. Strasser, M. Rooker, G. Ebenhofer, I. Hegny, M. Wenger, C. K. Sünder, A. Martel, and A. Valentini, "Multi-domain model-driven design of industrial automation and control systems," in *Proceedings IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2008.

[X] I. Terzic, M. Merdan, A. Zoitl, and I. Hegny, "Modular assembly machine - ontology based concept," in *Proceedings IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE Press, 2008.

[XI] I. Hegny, A. Zoitl, and W. Lepuschitz, "Integration of simulation in the development process of distributed IEC 61499 control applications," in *Proceedings 2009 IEEE International Conference on Industrial Technology*, 2009.

[XII] M. Merdan, W. Lepuschitz, I. Hegny, and G. Koppensteiner, "Application of a communication interface between agents and the low level control," in *Proceedings of the 4th International Conference on Autonomous Robots and Agents*, 2009.

[XIII] G. Koppensteiner, M. Merdan, W. Lepuschitz, and I. Hegny, "Hybrid based approach for fault tolerance in a multi-agent system," in *Proceedings of the 2009 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, 2009.

[XIV] T. Strasser, M. Rooker, I. Hegny, M. Wenger, A. Zoitl, L. Ferrarini, A. Dede, and M. Colla, "A research roadmap for model-driven design of embedded systems for automation components," in *INDIN 2009 7th International Conference on Industrial Informatics*, 2009.

[XV] A. Zoitl, I. Hegny, and A. Schimmel, "Utilizing binary XML respresentations for improving the performance of the IEC 61499 configuration interface," in *INDIN 2009 7th International Conference on Industrial Informatics*. IEEE, 2009.

[XVI] I. Hegny and A. Zoitl, "Component-based simulation framework for production systems," in *Proceedings 2010 IEEE International Conference on Industrial Technology*, 2010.

[XVII] T. Strasser, G. Ebenhofer, M. Rooker, and I. Hegny, "Domain-specific design of industrial automation and control systems: The MEDEIA approach," in *Preprints 10th IFAC Workshop on Intelligent Manufacturing Systems (IMS 10)*, 2010.

[XVIII] G. Koppensteiner, M. Merdan, I. Hegny, W. Lepuschitz, S. Auer, and B. Groessing, "Deployment of an ontology-based agent architecture on a controller," in *Proceedings INDIN 2010 - 8th IEEE International Conference on Industrial Informatics*. IEEE Conference Proceedings, 2010.

[XIX] I. Hegny, M. Wenger, and A. Zoitl, "IEC 61499 based simulation framework for model-driven production systems development," in *Proceedings IEEE Emerging Technologies and Factory Automation (ETFA 2010)*, 2010.

[XX] M. Wenger, M. Melik-Merkumians, I. Hegny, R. Hametner, and A. Zoitl, "Utilizing IEC 61499 in an MDA control application development approach," in *2011 IEEE Conference on Automation, Science and Engineering, August 24-27,2011, Trieste, Italy, Proceedings*, 2011, pp. 495–500.

[XXI] I. Hegny, T. Strasser, M. Melik-Merkumians, M. Wenger, and A. Zoitl, "Towards an increased reusability of distributed control applications modeled in IEC 61499," in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies and Factory Automation (ETFA 2012)*, 2012.

[XXII] M. Melik-Merkumians, T. Baier, M. Steinegger, W. Lepuschitz, I. Hegny, and A. Zoitl, "Towards OPC UA as portable SOA middleware between control software and external added value applications," in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies and Factory Automation (ETFA 2012)*, 2012.

[XXIII] F. Andren, T. Strasser, A. Zoitl, and I. Hegny, "A reconfigurable communication gateway for distributed embedded control systems," in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, 2012, pp. 3700–3706.

[XXIV] M. Merdan, A. Prostejovsky, I. Hegny, W. Lepuschitz, F. Andren, and T. Strasser, "Power distribution control using multi-agent systems," in *Recent Advances in Robotics and Automation*, G. Sen Gupta, D. Bailey, S. Demidenko, and D. Carnegie, Eds. Springer-Verlag, Berlin-Heidelberg, 2013, pp. 323–333.

Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt wurde. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Die Arbeit wurde bisher weder im In– noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Zwentendorf an der Donau, im Februar 2014

Ingo Hegny